Reactive Index Replication for Distributed Search Engines

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

Vincent Leroy
Yahoo! Research
Barcelona, Spain
leroy@yahoo-inc.com

Matthieu Morel
Yahoo! Research
Barcelona, Spain
matthieu@yahoo-inc.com

ABSTRACT

Distributed search engines comprise multiple sites deployed across geographically distant regions, each site being specialized to serve the queries of local users. When a search site cannot accurately compute the results of a query, it must forward the query to other sites. This paper considers the problem of selecting the documents indexed by each site focusing on *replication* to increase the fraction of queries processed locally. We propose RIP, an algorithm for replicating documents and posting lists that is practical and has two important features. RIP evaluates user interests in an online fashion and uses only local data of a site. Being an online approach simplifies the operational complexity, while locality enables higher performance when processing queries and documents. The decision procedure, on top of being online and local, incorporates document popularity and user queries, which is critical when assuming a replication budget for each site. Having a replication budget reflects the hardware constraints of any given site. We evaluate RIP against the approach of replicating popular documents statically, and show that we achieve significant gains, while having the additional benefit of supporting incremental indexes.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Performance

Keywords

Multi-site web search engine, distributed index, replication

1. INTRODUCTION

Distributed search engines aim at horizontal scalability for Web search [2, 6]. The search engine is distributed over several search sites deployed in (smaller) data centers spread across the world. Each search site only indexes a fraction of the documents, and receives the queries of the users in its region. When a query cannot be answered locally, it is forwarded to the other sites in order to compute accurate responses [2, 8]. Distributed search engines share the workload among several sites. Thus, deploying an extra site in a new data center increases the capacity of the search engine.

Designing an efficient distributed search engine is a challenging task. Traditional search engine algorithms rely on all the resources being accessed within a single data center. However, in the case of distributed search, the index is split among several distant locations. Accessing data in another search site translates into a higher response time due to network latency. Ideally, a site processes most queries locally to offer low response time and avoid the cost of processing the query in several sites. To enable this property, a careful selection of the documents to index on each site becomes necessary to guarantee that most frequently requested documents can be retrieved locally. A trivial solution is of course to replicate all documents across all sites. This solution, however desirable from a latency perspective, induces a high utilization of computer resources. Consequently, an important goal is to reduce resource utilization while providing low latency to end users through local query processing.

Brefeld *et al.* use machine-learning techniques to assign each document to one or multiple sites [4]. Their approach, however, does not consider the popularity of documents or a resource budget for each site. The popularity of documents becomes important when having to selectively drop documents due to resource constraints. Following a different approach, Blanco *et al.* propose to assign each document to a single site, and later rely on user activity to proceed to documents replication [3]. Their method relies on terms distribution and cache invalidations of a document to predict at which search site it is most likely to be requested. However, they do not propose any replication strategy.

Upon receiving a query, a search site processes it against its local index and then estimates whether other sites may have better documents, in which case the query is forwarded. Several heuristics have been developed for estimating the quality of documents in different sites. They rely on some partial knowledge of the content of the other search sites, either per-term score thresholds [2], or the scores of previously processed queries [8]. These algorithms are conservative: they can generate false positives, but no false negatives. Hence, the results computed by the distributed search engines are always the same as the ones generated by a single-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGIR'12, August 12–16, 2012, Portland, Oregon, USA.
Copyright 2012 ACM 978-1-4503-1472-5/12/08 ...$10.00.

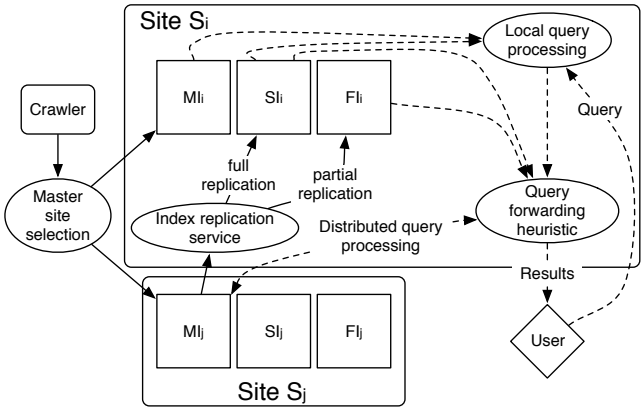

Figure 1: Architecture of a distributed search engine

site search engine. While these two approaches achieve a reasonable precision, they only target static indexes and require offline computation.

Contributions. We make the following contributions in this paper:

- We propose RIP (Reactive Indexing Protocol), a practical algorithm to selectively replicate documents and posting lists across sites in a distributed search engine. The algorithm makes decisions online and locally;
- RIP uses document popularity and user queries to select documents to replicate. Such information is important to ensure locality while respecting resource constraints;
- We evaluate the algorithm and compare it against a baseline that replicates documents statically, as well as a reactive documents replication algorithm. We show that RIP increases query locality by 23% while only increasing the index size of each site by 14%.

Roadmap. The remainder of this paper is organized as follows. We describe the architecture of the distributed search engine in Section 2. In Section 3 we present RIP, and evaluate its performance in Section 4. Finally, we review related work in Section 5 and conclude with Section 6.

2. ARCHITECTURE

We consider a distributed search engine comprising a set of search sites $\mathcal{S}$ deployed across geographically different regions. The collective of the sites forming the search engine indexes a document collection $\mathcal{D}$. We present a global view of the architecture of the distributed search engine in Figure 1, and describe the elements composing it throughout this section.

2.1 Traditional search engine architecture

The architecture of a search engine is typically divided into three components. The *crawler* fetches documents from the Web and discovers new content by following hypertext links. The *indexer* processes $\mathcal{D}$, the collection of documents, to generate an inverted index. For each term t present in the collection, the inverted index contains a posting list, *i.e.*, a list of the documents that contain t. A popular technique for implementing an index is incremental indexing. Incremental indexing enables the addition, deletion, or update of indexed documents without fully regenerating the index. This feature is particularly important in the case of large scale Web search engines where the cost of regenerating the full index to update frequently modified Web pages (*e.g.* news) is prohibitive. Incremental indexes offer a good trade-off between the freshness of the documents and the processing performance. Hence, in this paper, we consider the case of an incremental indexer. The *query processor* receives user queries and processes them against the index. Historically, commercial Web search engines have preferred conjunctive query processing [14]: a document has to contain all the terms in the query to be in the result set. To obtain the result set, the query processor computes the intersection of the posting lists of the query terms, evaluating the scores of the documents using a ranking function. The k (typically 10) results with the highest scores are returned to the user.

2.2 Assignment of documents to sites

In a distributed search engine, each search site has its own index and processes the queries of the users in its region. It is therefore important to carefully select documents to index in each site to tailor the data structures of the sites for their users. For a short response time, a search site must be able to process most of the queries it receives using its local index alone. Indexing documents that are popular in a region enables locality. However, it is also important to limit the number of documents indexed at each site. The query processing latency in fact increases with the size of the index [5]. Furthermore, replicating a large fraction of the documents at each site limits the scalability of the search engine. In this work, we assume that each search site has a fixed index capacity, either due to limited resources, or arbitrarily chosen to reduce the processing time. We express this limit using number of postings, *i.e.*, the sum of the length of all the posting lists in the index.

To the extent of our knowledge, two approaches have been developed to compute an assignment of documents to search sites. The first one relies on machine learning algorithms to analyze new documents upon their discovery and assigns them to one or more search sites [4]. The second one assigns each document to a single *master site* [3]. This assignment results in a minimal index, with each document indexed in a single location. The master site of a document is responsible for maintaining it in its index, which guarantees that the distributed search engine has the same recall as a centralized implementation. Our work builds up on the master selection approach [3]. We assume the existence of an algorithm that assigns each document to a single search site. Hence, the search site S_i has a master index MI_i which indexes all the documents S_i is the master of. Typically, this index represents the majority of S_i's index capacity.

The popularity of Web pages typically follows a power law: while most of the Web pages are unpopular, a few of them are requested very frequently. A Web page might present high locality, being popular in a single region, or be popular across many regions. Distributed search engines work better in a context where documents have a strong locality. Indeed, this means that each document only needs to be indexed by the search site located in the region where it is popular. Fortunately, a large fraction of the popular Web pages exhibit a high divergence in their popularity across regions [3]. Nevertheless, there are still documents which are popular across region boundaries and are requested by users

of different search sites. The master selection algorithm, ensures that each document is indexed by its master site.

In this paper, we propose RIP, a Reactive Indexing Protocol. RIP uses part of the remaining index capacity of each search site to replicate documents that are frequently requested locally, but were assigned to a different search site by the master selection algorithm. Contrary to the master selection algorithm, which only relies on the content of the document, RIP is *reactive*: it analyzes the behavior of the users at each site to dynamically adjust replication decisions. These fully replicated documents form the shadow index, denoted SI_i for the site S_i.

2.3 Distributed query processing

In a distributed search engine, a search site indexes locally only a fraction of the documents. To preserve the quality of results, a distributed search engine must generate the same results as a centralized implementation. A query submitted to a site S_i must be evaluated on the full set of documents $\mathcal{D}$, whether they are indexed locally (MI_i and SI_i) or not. So far, the main approach to executing queries in a distributed search engine relies upon query forwarding [2, 8]. When processing a query, a search site first computes results using its local index, and then relies on a forwarding heuristic to determine whether another site may be able to provide higher quality results. If there are such sites according to the evaluation of the heuristic, then the query is forwarded to the relevant search sites for further processing. Finally, the results are then merged and returned to the user.

The forwarding heuristic is conservative with respect to query forwarding, and it can generate false positives, but no false negatives. As already mentioned in Section 2.2, it is preferable to answer a query locally, since it reduces the response time. It is therefore important to devise an accurate forwarding heuristic to reduce the forwarding rate.

Existing forwarding heuristics [2, 8] leverage properties of the search engine ranking function to compute an upper bound on the score of documents which are not indexed locally. This ranking function $s(d|q)$, presented on Figure 2, was introduced by Baeza-Yates *et al.* [2]. The score of a document d is computed by averaging partial scores over the terms of the query q. A document d has a quality score, expressed by $f(d)$, and a relevance score for a term t, computed by $g(d|t)$. The parameters w_f and w_g weight quality and relevance respectively. The partial score of a document d for a term t, expressed as $r(d|t)$, is typically maintained in the posting lists of the index to improve query evaluation performance. Note that distributed search engines are compatible with more complex ranking functions, including positional features, machine learning and diversification. In these cases, the $s(d|q)$ ranking function is used in the first phase of a two-phase ranking [9], which ensures that relevant documents will be known locally before the execution of the second phase of the ranking.

The heuristic we propose here supports the same ranking function. A site S_i may partially replicate the posting lists of the master indexes of other sites in order to estimate the need to forward queries. This forwarding index is designated as FI_i. The posting lists of forwarding indexes are ordered by partial score. For a given term t, S_i replicates the list of documents that have the highest partial score $r(d|t)$. A document whose master is S_j and that contains several terms may only be indexed in one of the posting lists of FI_i, provided its other partial scores are low.

$$s(d|q) = w_f f(d) + \frac{w_g}{|q|} \sum_{i=1}^{|q|} g(d|t_i)$$
$$r(d|t) = w_f f(d) + w_g g(d|t)$$
$$s(d|q) = \frac{1}{|q|} \sum_{i=1}^{|q|} r(d|t_i)$$

Figure 2: Ranking function

2.4 Index structure

In this section, we summarize the different elements of the index of a search site. The index of a site S_i is logically divided into three components:

Master index MI_i: It contains the documents that were assigned to S_i by the master selection algorithm.

Shadow index SI_i: It contains documents that are fully replicated by S_i. They were assigned to one of the other search sites by the master selection algorithm, but are replicated to improve the query processing locality.

Forwarding index FI_i: It contains partial information about documents assigned to the other search sites. For a given term t, the posting list associated to t in FI_i contains the list of documents that have the highest partial scores for t. Documents having high partial scores are also likely to have a high popularity. Hence, SI_i and FI_i may overlap.

These logical indexes represent the data available to S_i for processing queries. The posting lists of search engines are, in most cases, ordered by document ID. This allows high compression rates, and is particularly efficient for conjunctive query processing [14]. In this work, we do not make any assumption on the layout of MI_i and SI_i. However, we present FI_i as an index sorted by impact. This means that the postings are ordered by partial scores. This assumption simplifies the description of the algorithm, but there is, in practice, no reason not to implement these three indexes as a single incremental index relying on document ID ordered posting lists.

3. REACTIVE INDEXING PROTOCOL

3.1 Problem definition

We consider the following problem. In a distributed search engine, each search site is assigned an index budget expressed as the maximum number of posting lists entries a site can accomodate. The collection of documents $\mathcal{D}$ is split across the search sites through an initial master index selection algorithm. The remaining index capacity of each site is freely used to replicate documents and posting lists of the other sites. A sequence of queries is submitted to each search site and the goal is to maximize the amount of queries that are answered locally, without query forwarding. Consequently, a search site updates its index as it processes queries. A search site modifies its index only between query executions, and it does so for a given query only after its results are returned to the user.

Let $res(q)$ be the set of the k documents obtaining the highest scores for the query q according the the search en-

gine's ranking function $s(d|q)$. A site S_i has to fulfill two conditions to answer q locally. First, the documents of $res(q)$ must all be indexed and copied locally. The search site needs a copy of the document data to generate the snippet presented on the results page, and also in the case that it uses using a two-phase ranking [9]. The search site also needs to be able to compute accurate scores for all the top-k results to display them properly ranked. This requirement can be expressed as follows:

$$\forall d \in res(q), d \in \mathrm{MI}_i \vee d \in \mathrm{SI}_i \ .$$

Second, S_i should be able to determine, using local data structures, that no other document could potentially score higher than the lowest score of the results:

$$\forall d \in (\mathcal{D} - res(q)), \forall e \in res(q), scBound(d|q) \leq s(e|q) \ ;$$

where $scBound(d|q)$ is the function that computes an upper bound on the score of the document d for the query q using only local information, *i.e.* the information from MI_i, SI_i and FI_i.

We estimate the future queries Q_i^f of S_i using Q_i, a recent query stream received by S_i. Thus, at any given point, we are trying to maximize the locality of the queries in Q_i to increase the probability that future queries will be answered locally. Let us focus on the first locality condition. The cost of replicating a document is equivalent to the number of posting list entries it requires in the index. To simplify the problem, suppose that all documents contain the same number of terms and therefore have the same cost. The problem we are trying to solve is a particular form of the knapsack problem. The objects we are selecting are queries. The utility of selecting a query is proportional to its frequency, while its cost is equal to the indexing of the results, as well as the partial information ensuring the quality of results.

Given that the search engine aims at serving the k best results for each query, we could make the simplifying assumption that all queries have the same cost: indexing k documents. However, even in this case, the complexity of the problem arises from the fact that many queries share results. Hence, the cost of a selection is not equal to the cost of each query, as some documents would be counted several times. The knapsack problem is NP-hard, but has greedy heuristics that perform reasonably well. In particular, the most common approach consists of selecting the objects in a decreasing order of $\frac{value}{cost}$. In our case, the cost of selecting a query depends on the previously selected ones; hence the costs should be re-evaluated at each step of the algorithm.

3.2 Practical approach

As presented in Section 3.1, a search site has to satisfy two conditions to answer a query locally. The first one is that the results should be indexed locally, and the second one is that the search site should have enough information to guarantee that no other document in $\mathcal{D}$ can score higher. The computation of an exact optimal solution is NP-hard, which leads us towards heuristic solutions. Moreover, we need to consider practical implementation constraints in the design of our algorithm.

This work targets search engines performing incremental indexing. The index of the search engine is regularly updated, and the algorithm must account for the presence of new documents. This eliminates the possibility of relying solely on an offline algorithm executed during the initial index generation. A practical solution should also use a minimal amount of computation and memory, so as to ensure that as many resources as possible are dedicated to query processing. Part of the computational cost of the problem presented in Section 3.1 arises from the fact that the benefit of replicating a document cannot be simply evaluated, it depends on the full selection of replicated documents. From this consideration, it would be tempting to devise a solution based on hypergraphs of documents and queries to model replication dependencies. However, given the scale of the document collections we consider, such data structures do not scale, both with respect to their memory consumption and the processing cost required to exploit them.

Inspired by previous work in Web caches [13], we propose a Reactive Indexing Protocol (RIP). Each search site engine monitors the queries of its users to gather local statistics about the frequency of terms and documents. Based on these observations, our algorithm evaluates the utility of replicating information. A search site S_i may either replicate documents, to ensure that the results are copied locally, or fragments of the posting lists of other sites, to increase its knowledge of their document collection and make more accurate query forwarding predictions by computing tighter score bounds. As introduced in Section 2.4, S_i indexes fully replicated documents in SI_i, while the replicated fragments of posting lists form FI_i.

3.3 Algorithm

Distributed search engines rely on query forwarding heuristics to determine whether a given query q should be evaluated on other search sites to improve the quality of the results. The role of the forwarding heuristic is to compute, for all documents $d \in \mathcal{D}$, a score upper bound $scBound(d|q)$. If the top-k documents are either not fully replicated locally, or cannot be clearly identified, the search engine decides to forward the query to the other sites to guarantee the quality of results. In this work, we introduce a new query forwarding heuristic and RIP, its associated index replication algorithm.

3.3.1 Forwarding heuristic

The forwarding heuristic we propose stems from the NRA top-k processing algorithms [11], with a few adaptations to deal with incomplete posting lists. In NRA, posting lists are sorted by impact and processed from top to bottom. NRA maintains a sorted heap of potential top-k results with upper and lower bounds on their scores. These bounds are updated as the processing progresses down the posting lists. As soon as the upper bound of the $(k+1)^{th}$ document is lower than the lower bound of the k^{th} document, the top-k results are identified and the algorithm terminates. In the worst case, NRA has to process the full posting lists, but, in most situations, it achieves significant performance gains and only processes a small fraction of the index. The forwarding heuristic performs a similar computation. It processes the query over the forwarding index FI_i and computes upper bounds on scores. The posting lists of the forwarding index are only partial, but they are continuous. A posting list replicated by S_i for a term t down to the score value v contains all the documents of $\mathcal{D}$ whose master is different from S_i and whose partial score $r(d|t)$ is higher than v. Therefore, for a given term, FI_i provides either an exact partial score, or an upper bound equal to the score of the last posting list entry. While processing, the forwarding heuristic

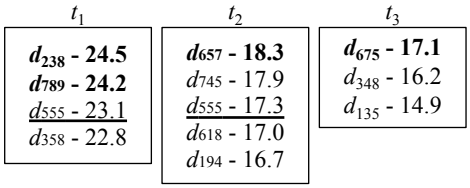

Figure 3: Forwarding heuristic on FI_i

ignores the documents present in the shadow index SI_i, as they are already evaluated by traditional query evaluation and are assigned a precise score.

We illustrate the forwarding algorithm with the example of Figure 3. The query of the user is "t_1, t_2, t_3", and the figure displays the posting lists of FI_i corresponding to those terms which the forwarding heuristic evaluates to decide whether the query should be forwarded. The top documents for t_1 are replicated in SI_i (in bold), so they do not need to be considered. The following document is d_{555}, so we know its exact partial score for this term. This document is also present in the posting list of t_2, so we will also find its exact partial score for t_2 as the top-k execution progresses. However, d_{555} is not represented in the posting list of t_3. The last known document of this posting list is d_{135}. As a consequence, we use its partial score as an upper bound of d_{555}'s partial score for t_3. Hence, the upper bound score computed for d_{555} is (23.1+17.3+14.9)/3=18.4. We can also compute a bound on the score of any document absent from these posting lists using the scores of the last entries (22.8, 16.7 and 14.9 in this example). Using FI_i, the forwarding heuristic computes the highest possible score for a document that is not indexed locally and compares it with the score of local documents (MI_i and SI_i).

As an optimization, when a posting list is fully replicated (*i.e.* replicated down to the 0 score), the forwarding heuristic leverages the conjunctive properties of the ranking function. Any document that is absent from this posting list can be ignored, as it cannot be part of the results.

3.3.2 Replication principle

The replication algorithm works as follows. For each term t, a site maintains two replication thresholds, expressed in partial score values: the document replication threshold td_t and the postings replication threshold tp_t. RIP reactively adjusts these thresholds using the activity of the local users to determine which documents and postings are replicated.

$$\forall d \in \mathcal{D}, r(d|t) \geq td_t \wedge master(d) \neq i \Rightarrow d \in SI_i$$

$$\forall d \in \mathcal{D}, r(d|t) \geq tp_t \wedge master(d) \neq i \Rightarrow d \in FI_i$$

For the example described on Figure 3, td_{t_1} is 24.2, while tp_{t_1} is 22.8. By lowering td_t, RIP decreases the highest scores associated to t for a non local document. Lowering tp_t decreases the lowest score associated to t in FI_i. Both these actions increase the information related to the term t and decrease the amount of query forwarding. However, their impact and cost can vary significantly. Fully replicating a document is costly, as it generates one posting entry per unique term in the document. On average, a Web page contains 250 unique terms [15], therefore replicating a document is 250 times more costly than replicating a posting entry. Given that the differences in partial scores between entries are, in most cases, higher among high quality documents, fully replicating a document often has a higher positive impact on query forwarding. RIP's objective is to achieve a good balance between documents and postings replication to use the replication budget as efficiently as possible.

After each query execution, RIP analyses the query results to determine which data should be replicated to ensure that, in the future, this query could be processed locally. Let w be the lowest score of the last document returned as a result for the query $t_1 \ldots t_{|q|}$. If the query only contains one term, then the replication operation is trivial, and the algorithm determines that td_{t_1} should be w. However, if the query contains several terms, then the algorithm has to decide whether it should replicate documents or posting lists. The algorithm we propose relies on a parameter α to balance the replication between documents and postings.

$$td_t = \alpha \times |q| \times w$$

$$tp_t = \frac{(1-\alpha)|q| \times w}{|q| - 1}$$

Using the scoring function $s(d|q)$, it is possible to verify that for all the documents present in a single posting list of FI_i, the forwarding heuristic has enough data to compute a score upper bound at most equal to w:

$$\forall t \in q, \frac{1}{|q|}\left(td_t + \sum_{u \in q - \{t\}} tp_u\right) = w$$

By definition, replicating document provides the corresponding postings, so $td_t \geq tp_t$. As a consequence, given that $|q| \geq 2$, $\alpha \geq 0.5$. When α is low, RIP favors replicating documents, which increases the probability of having query results in the local index. However, the forwarding heuristic has less information to ensure that these local results are optimal. On the contrary, a high value of α provides a very accurate forwarding heuristic, but fewer replicated documents.

In practice, some of the documents are present in several posting lists. Hence, they have precise values for several terms, and their score estimations may exceed w. The results of the query, for instance, will be present in all the posting lists matching the query, and will generate scores higher than w. Similarly, other documents present in at least 2 posting lists, such as d_{555} in Figure 3, may have high upper bounds on their score and could trigger the query forwarding mechanism. When these documents are not part of the query results, query forwarding is unnecessary. In order to avoid these cases of false positives, RIP identifies these documents and fully indexes them in SI_i.

3.3.3 Practical algorithm using blocks

RIP needs to estimate the amount of documents or postings a replication decision represents before deciding whether it should be applied or not. Furthermore, taking replication decisions at the level of a single posting may lead to unstable results and generate a high overhead.

Each search site estimates loosely the score distribution for each term by regularly probing the other search sites. This data structure is comprised of blocks, and is illustrated in Figure 4. A block constitutes a unit of replication identified by its index as well as its score bounds. This information

block index/size	t_4	t_5
0/10	**Upper = 15.7, Lower = 12.7**	*Upper = 17.1, Lower = 15.3*
1/20	**Upper = 12.7, Lower = 9.8**	*Upper = 15.3, Lower = 13.7*
2/40	Upper = 9.8, Lower = 7.3	*Upper = 13.7, Lower = 6.4*
3/80	Upper = 7.3, Lower = 4.8	Upper = 6.4, Lower = 1.8

Figure 4: Blocks replication ($k = 10$)

is not required to be perfect, and can be obtained through sampling. The first block has size k, and the size of the following blocks increases exponentially, using a power of 2.

We adapt RIP to apply the replication thresholds td_t and tp_t on blocks of documents and postings instead of single elements. Figure 4 presents the computation of the thresholds for the query "t_4, t_5". The lowest score $w = 8.5$, and $\alpha = 0.6$. RIP determines that td_{t_4} should be $0.6 \times 2 \times 8.5 = 10.2$. Therefore, the first 2 blocks of documents should be replicated (in bold), and dt_{t_4} becomes 9.8, once adjusted to the block limit. tp_{t_5} is evaluated to be $(1-0.6)\times 2\times 8.5/(2-1) = 6.8$, which means that 3 blocks of postings should be replicated (in italics) and tp_{t_5} is adjusted to 6.4. This operation is then repeated for td_{t_5} and tp_{t_4}.

The usage of blocks brings several benefits to RIP. It materializes a small number of well defined replication bounds, which favors clear replication decisions while lowering the amount of memory required to maintain them. In addition, it ensures the continuity of replicated data. A replication decision caused by a given query may, as a side effect, trigger the replication of blocks that contain documents useful for other future queries.

3.3.4 Space management

RIP does not directly proceed to the replication of data. Instead, it maintains counters about the number of times a particular piece of information was determined to be useful. We refer to these counters as the temperature of the data. The higher the temperature, the more useful it is. The temperature values are used to determine which data should be replicated within the budget constraint. As explained previously, depending on the data type, the cost of the replication varies. Given that a document has on average 250 distinct terms, the cost of replicating a document is on average 250, the cost of replicating the documents of a block of size n is on average $250n$, and the cost of replicating only the postings of this block is exactly n.

We rely on the cubic selection scheme [13] to decide which elements are selected for replication. This approach was initially designed to cache Web objects. Hence, in this context, the size of an object is always precisely known, and there are no dependencies between objects. In our case, the size of an object is first estimated, and then corrected when a replication decision is actually taken. For instance, RIP first assigns a cost of 250 to a document, and then corrects it upon receiving the content of the document. Similarly, evicting a block whose documents are replicated does not necessarily free $250n$, as some of the documents may remain replicated due to other decisions (*e.g.* replicated blocks of other terms). We also ensure that the continuity of block replication is maintained. It is impossible to evict a block without evicting all the following blocks.

3.3.5 Dealing with incremental indexing

Web search engines are often designed to support incremental indexing. This feature is particularly used for popular Web pages that are frequently modified, such as news Websites. Replicating information in several locations poses the problem of data consistency. If a search site does not take into account a new version of a Web page, it will serve stale results to its users.

To ensure that index updates are propagated, a search site keeps track of which other sites replicate the documents it is the master of. In addition, for each term, it maintains the documents and postings replication thresholds of each one of the other sites. When the crawler sends a new version of a document to its master site S_i, MI_i is updated and all the sites replicating any stale data are notified.

4. EVALUATION

To evaluate RIP's efficiency, we simulate a distributed search engine configuration consisting of five search sites $\mathcal{S} = \{S_1 \dots S_5\}$. We first compute optimal query results through a centralized search engine indexing all documents, and then evaluate each site's ability to generate these results using its master index and the data it replicates. Each search site is associated with a query log originating from neighboring countries and collected from the front-end of a commercial search engine. In total, we sample $7,023,102$ consecutive queries, and split them chronologically into a training set and a testing set of equal size. The collection of documents consists of $31,599,910$ Web pages randomly sampled from the index of the same search engine.

We rely on the distribution of terms in queries and documents to assign each document to a master site (KL-Q feature), as in the work of Blanco *et al.* [3]. This process creates, for each site, the master index MI, and represents the initial assignment of documents to sites in our evaluation. In this work, the master index creation process is fixed, and these indexes remain constant throughout the experiments. We evaluate RIP's ability to generate SI and FI, and to reduce the query forwarding rate.

Commercial search engines often cache query results to avoid re-evaluating repeated queries. To make our experiment more realistic, we assume that the search engine implements a results cache with a Time-To-Live (TTL): cached results are only served if their TTL has not expired. We use a TTL value of 2 hours. For incremental indexes, it is possible for a results cache to return stale results, which happens when the result set does not reflect accurately the current state of the index. In our setup, the TTL of the cache is 2 hours. Such a TTL value is moderately aggressive: the staleness of the results will never exceed 2 hours.

4.1 Search results diversity

Blanco *et al.* [3] observe that most documents exhibit a high divergence of popularity among the different search sites. They are very popular in a given region, but are rarely requested at other search sites. We confirm this observation by evaluating how the popularity ranking of documents differs across search sites. We execute the training queries on the search engine and count how often each document is returned as part of the top-10 results.

In Figure 5, we present the similarity between the list of documents that are most popular at each individual search site, and the documents that are globally popular, among

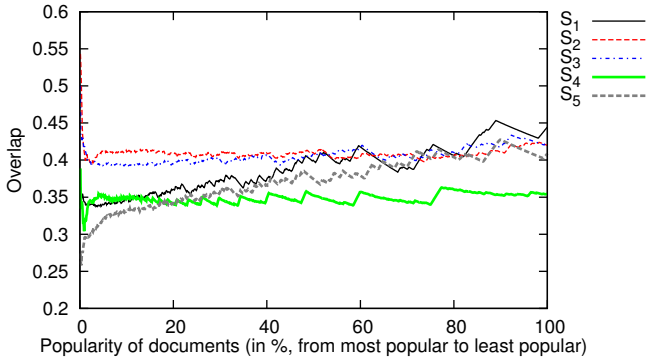

Figure 5: Similarity between documents locally popular and documents globally popular

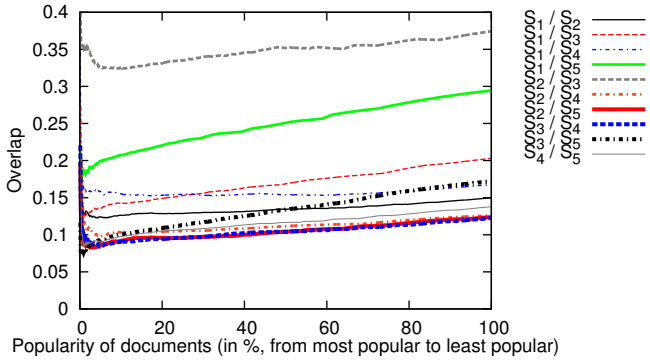

Figure 6: Similarity between locally popular documents at 2 sites

all the search sites taken together. On average, the set of the x documents most popular at a given site and the set of the x documents most popular considering the activity of all sites globally only overlap by 40%. This observation clearly illustrates how users from different regions have diverse interests. This argues in favor of an index replication policy that relies on local user activity and optimizes the replicated data for each site individually.

The comparison of popular documents between pairs of sites, presented in Figure 6, shows even more differences. The pair of sites exhibiting the highest similarity is under 40%. This similarity is due to the language used in queries, as those two regions share the same regional language. Note that the similarity of the very few most popular documents is typically higher. This is due to very few documents being popular across different regions.

4.2 Cache impact

The results cache generates a hit when identical queries are submitted to the search engine within its TTL interval. Longer queries typically present lower frequency compared to short ones [1], and therefore the results cache affects them differently. In our experiments, the average hit rate of the results cache is 48.4%. Figure 7 illustrates the distribution of queries depending on their length. Queries of length 2 represent the largest fraction of the workload, followed by single term queries. As expected, the hit rate of the cache drops

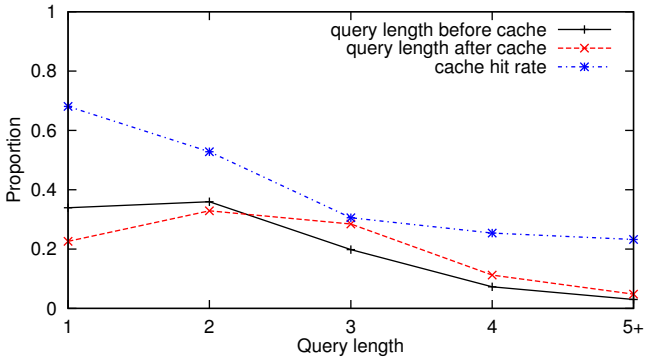

Figure 7: Queries distribution and cache efficiency

as we increase the length of queries; long queries are more likely to be unique. Hence, the workload of a search engine is often dominated by the processing of longer queries. This observation significantly hardens the task of the forwarding heuristic. Indeed, long queries involve more posting lists, and generally have results whose partial scores are lower: they require more replicated data to be indexed locally.

4.3 Replication and query forwarding

We evaluate RIP and its query forwarding heuristic by running queries in our logs against the collection of documents. First, we warm up the replication algorithm and the cache using the training queries. Next, we process the testing queries to evaluate the amount of query forwarding. For these experiments, we use the forwarding heuristic of Section 3.3.1 and one of the following replication schemes:

Static global documents replication (SDR): Using the training queries, we determine which documents are globally popular and replicate this static set across all sites [2, 8]. In this case, FI contains an upper bound per term for non replicated documents, which corresponds to the thresholds of Baeza-Yates *et al.* [2].

Reactive documents replication (RDR): Each search site reactively determines which documents are most frequently part of the results of their users and replicate the most popular ones. This setup computes a different set of replicated documents for each site to match the activities of their users. As with SDR, FI contains one bound per term, dynamically adjusted to reflect document replication.

Reactive Indexing Protocol (RIP): Each site reactively replicates blocks of documents and posting lists, as well as individual documents when they generate false positives. This is the approach we describe in Section 3.3.3. Each site replicates data matching the needs of its users, while preserving the continuity of information in posting lists.

Overall performance.

We evaluate the three different replication schemes using different replication budgets and present the results in Figure 8. The budget is represented as the maximum number of documents replicated, and we assume, as explained in Section 3.3.2, that 250 individual posting list entries use the same space as a fully indexed document. The results clearly indicate that the two algorithms based on local information outperform the static documents replication us-

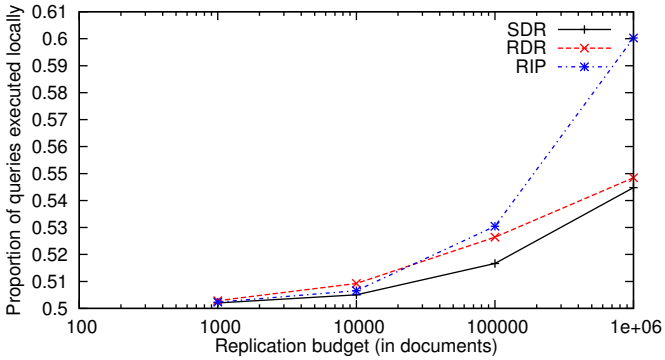

Figure 8: Query locality wrt replication budget

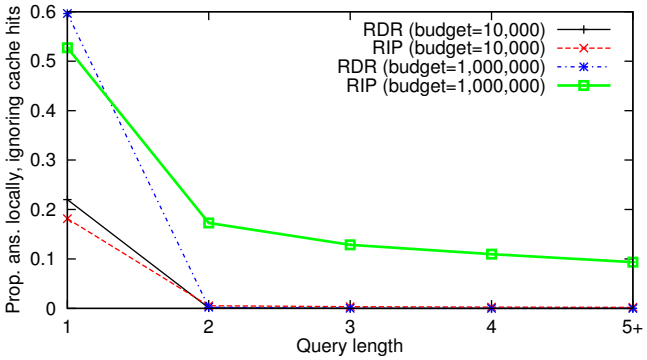

Figure 9: Query locality wrt query length

ing global statistics. Indeed, as observed in Section 4.1, the users of each site are interested in different documents.

For a low replication budget, below 100,000, we observe that simply replicating the results of the queries is more efficient than replicating blocks of documents and posting lists. However, as the budget increases, the blocks replication scheme clearly outperforms the replication of individual documents. This difference grows with the amount of space dedicated to replication. With a replication budget of 1,000,000 documents, each search site has an average indexing capacity of 7,119,982 documents (22.5% of the total collection), which represents an overhead of 14% over a setup without any replication. In this configuration, RIP raises the amount of queries processed locally by 23%, while RDR raises it by 13%. Note that the α parameter of RIP, used to compute replication thresholds, is set to 0.6. In practice, any value between 0.55 and 0.65 obtains good performance, above 59.7% with a budget of 1,000,000.

Detailed performance analysis.

We detail the performance of RDR and RIP in Figure 9. With a small replication budget, it is most efficient to focus replication on single term queries. They only require little replicated data to be answered locally, as the results are simply the documents with the highest scores for the query term. RDR performs well for these easy queries. Given that one-term queries are less likely to be unique, the temperature of their results increases over time and they are replicated. However, when the replication budget increases, it becomes more interesting to also replicate data for longer queries. The results show that RDR is unable to answer these queries, even with a large budget. The documents that are part of the results may be replicated. However, given that the corresponding posting lists are not replicated, the search engine is unable to ensure that the query results are optimal, and the forwarding heuristic returns a false positive. RIP can compute low thresholds, even for longer queries, and is able to answer locally over 10% of long queries by replicating continuous blocks of documents and postings.

Query replication cost analysis.

Figure 10 presents the position of the query results in the posting list blocks of RIP's forwarding index, depending on the query length. For a one term query, the result set comprises the documents with the highest partial scores for

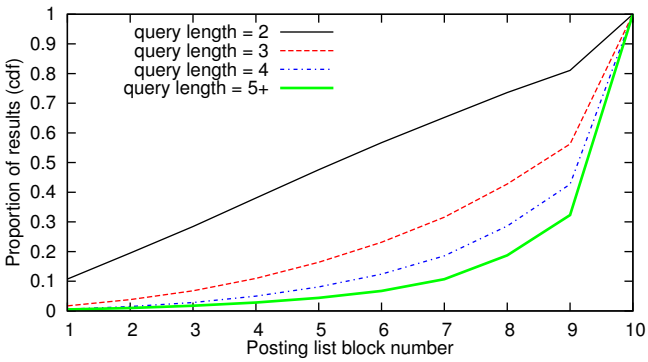

Figure 10: Depth of results in posting lists

the term. Hence, they are all located in the first posting list block, which is a small amount of information for RIP to replicate to answer these queries correctly. However, as the length of the query increases, the matching documents are less frequent, due to the conjunctive nature of the query processing. As a consequence, they are located in deeper blocks, and require more replicated information to enable local processing. For example, 67% of the results of 5-term queries are located in the 10^{th} block. Given that the size of blocks is a power of two, this data is costly to replicate, making forwarding more frequent for long queries.

New queries.

The replication algorithms rely on previous queries to compute a replication scheme and increase the probability of answering future queries locally. When a query is repeated, it can be answered by the results cache, if it falls within the TTL, or by the data replicated upon the first occurrence of the query. New queries however are more challenging. We examine the query processing locality for new queries on Figure 11, with a replication budget of 1,000,000.

SDR is particularly efficient at processing new one-term queries locally, since it benefits from document popularity information from all search sites. A query that is processed for the first time in a site might have been present at another site during the training period. Consequently, the static replication has included it in the computation of the list of replicated documents.

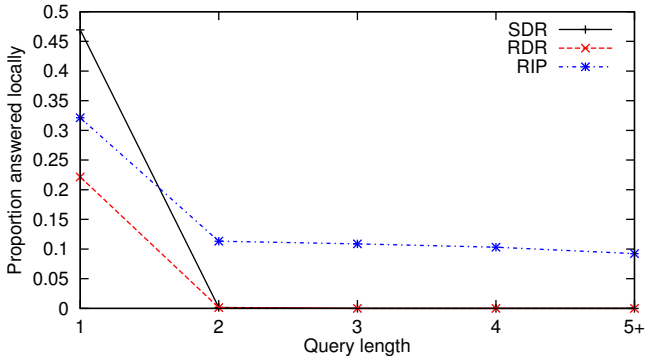

Figure 11: Query forwarding rate for new queries

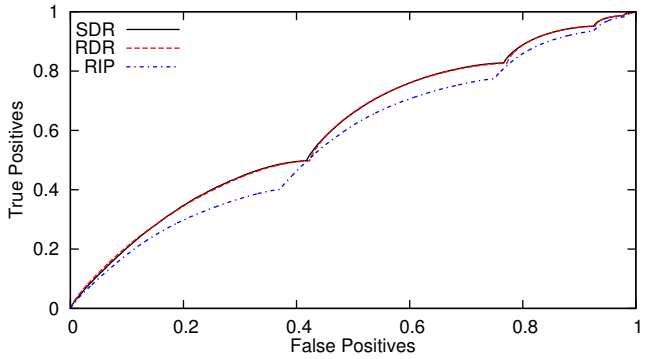

Figure 12: Forwarding ROC curve

RDR has higher overall performance than SDR, since it targets the set of replicated documents for each site. However, since it only relies on the queries processed at a given site to build the list of replicated documents, the performance for new queries is low. New queries can only be processed locally if their data has been replicated due to previous distinct queries executed at this site. As RDR only replicates the documents in result sets, it is unlikely that the site has replicated all the results of the new query and the data necessary to ensure that local results are sufficient.

RIP performs well for new queries. Although it only relies on local knowledge, as it is the case for RDR, the block replication pattern favors new queries. Instead of precisely replicating the documents answering a particular query, RIP transfers blocks of data, which contain additional documents and postings related to the query terms. When a new query arrives, the algorithm is more likely to have replicated a sufficient amount of information for each of the query terms, which increases the probability of processing it locally.

False positives analysis.

Our forwarding heuristic always forwards a query if there another search site may improve its results. As a consequence, the forwarding heuristic does not generate false negatives. For some queries, the search site has the query results in its index but forwards the query nevertheless, because it cannot prove that those are the best results. These cases constitute false positives, as forwarding the query does not modify its results. Considering a setup with a replication budget of 1,000,000, the proportion of false positives among the forwarding decisions is 51%, 53% and 46% for SDR, RDR, and RIP respectively.

Allowing the search engine to return non-optimal results by allowing false negatives reduces the query forwarding rate. The forwarding heuristic computes a score that corresponds to the limits of its knowledge. We use this score, as well as the fact that long queries are more difficult to predict, to rank forwarding decisions. We display, on Figure 12, the distribution of true and false positives (ROC curve [12]). As one-term queries only involve one posting list, making it impossible to generate a false positive, we ignore them in this experiment. The remaining query lengths generate distinctive curve fragments. These fragments remain relatively close to a diagonal, which means that separating true and false positives using solely the knowledge score and query length is difficult. Indeed, scores can vary significantly depending on the query term, and cannot be easily compared. Note that the curve representing RIP is bellow the one of the other algorithms. This is because RIP presents an overall higher performance, and therefore forwards fewer queries. The remaining false positives are therefore harder to detect.

5. RELATED WORK

The problem of assigning documents to sites in distributed search engine has been studied in previous work, and two main approaches have been developed to assign documents to sites. Baeza-Yates *et al.* propose to replicate a set of global, high quality documents to all search sites [2]; Cambazoglu *et al.* follow the same approach [8]. As shown in Section 4.2, the documents users are interested in significantly differ across regions. Thus, it is more efficient to perform fine grain replication, and select different replicated documents for each search site. Our experiments presented in Section 4.3 show that per-site replication algorithms systematically outperform global replication decisions.

Brefeld *et al.* propose to use machine-learning techniques to statically assign documents to search sites [4]. This approach optimizes the replication of documents for each site, but it relies on attributes, such as the language of the document, which are weakly correlated with its popularity. Contrary to this approach, the algorithm we propose is reactive, and only replicates documents when it observes that users actually request them. The algorithm we propose also explicitly prioritizes pieces of replicated data depending on their size and temperature. The approach of Brefeld *et al.* relies on an algorithm that simply assigns documents to search sites, and it does not enable an operator to vary the indexing capacity.

Existing query forwarding algorithms rely on the computation of upper bounds to estimate the score of documents that are not indexed locally. Baeza-Yates *et al.* [2] compute a bound for each term and each search site. Cambazoglu et al. [8] refine this approach and show that maintaining bounds on pairs of terms more precise estimates of scores and can reduce the amount of query forwarding. In the case of a static index, these bounds can easily be computed during the generation of the index, and do not vary at runtime. However, when the index and the set of replicated documents are dynamically selected, these bounds need to be updated. This process can become particularly costly when

each site keeps track of many bounds, which is the case in the algorithm of Cambazoglu *et al.* [8]. In that case, one possibility is to maintain these bounds lazily using a sliding window scheme. Yet, it reduces the effectiveness of the results cache, since results computed using these bounds will be time-stamped with the oldest bound used during their computation. The approach we propose explicitly maintains two bounds per term, which is a reasonable trade-off between their accuracy and the maintenance cost.

The main difference between the algorithms we propose and previous work [7, 8] is the interaction between the replication algorithm and the forwarding heuristic. While other algorithms focus on replicating popular documents, our approach also maintains single posting lists elements to ensure the continuity of the information on the scores for a given term. As there is no "hole" in the forwarding index, the upper bounds for each term are lower. In addition, by replicating postings further, we efficiently lower the bounds computed for longer queries. Overall, this significantly reduces the amount of query forwarding.

Ding and Suel [10] developed an algorithm for fast top-k processing on document ID-ordered posting lists. By maintaining upper bounds on the partial scores of compressed posting list segments, they significantly reduce both the amount of computation and the processing time. Our approach for block replication could be adapted to replicate document ID-ordered blocks using these thresholds. However, this would lead to a significant overhead for storing the index, since documents obtaining low scores would also be replicated if they belong to a segment with a high threshold. Consequently, we opted for strictly following the order of partial scores to replicate information.

6. CONCLUSION

We propose RIP, a Reactive Indexing Protocol for distributed search engines. With RIP, the search engine initially indexes each document on a single master site, and monitors the local user activity of each site to generate an index replication scheme. Our scheme replicates documents and fragments of posting lists. RIP enables a significant reduction of the amount of query forwarding between search sites, and consequently, of query processing latency. We show experimentally that, by making local decisions, RIP significantly outperforms previous replication strategies based on global information: in a 5-site setup, when each site has an index capacity of 22.5% of the documents collection, 60% of the queries are processed locally. We also show that replicating fragments of posting lists allows the query forwarding heuristic to compute lower score thresholds on unknown documents, which increases performance. Finally, RIP has by design the additional benefit of being an online approach, supporting both incremental indexing and a precise index capacity configuration, which simplifies operations in a production environment.

Acknowledgments

This work has been partially supported by the COAST project (ICT-248036), funded by the European Community. The authors have been also supported by the INNCORPORA - Torres Quevedo Program from the Spanish Ministry of Science and Innovation, co-funded by the European Social Fund.

7. REFERENCES

[1] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 183–190, 2007.

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

[3] R. Blanco, B. B. Cambazoglu, F. P. Junqueira, I. Kelly, and V. Leroy. Assigning documents to master sites in distributed search. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management*, pages 67–76, 2011.

[4] U. Brefeld, B. B. Cambazoglu, and F. P. Junqueira. Document assignment in multi-site search engines. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, pages 575–584, 2011.

[5] E. Brewer. Lessons from giant-scale services. *Internet Computing, IEEE*, 5(4):46–55, 2001.

[6] B. B. Cambazoglu, V. Plachouras, and R. Baeza-Yates. Quantifying performance and quality gains in distributed web search engines. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 411–418, 2009.

[7] B. B. Cambazoglu, V. Plachouras, F. Junqueira, and L. Telloli. On the feasibility of geographically distributed web crawling. In *Proceedings of the 3rd International Conference on Scalable Information Systems*, pages 31:1–31:10, 2008.

[8] B. B. Cambazoglu, E. Varol, E. Kayaaslan, C. Aykanat, and R. Baeza-Yates. Query forwarding in geographically distributed search engines. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 90–97, 2010.

[9] B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 411–420, 2010.

[10] S. Ding and T. Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 993–1002, 2011.

[11] I. Ilyas, G. Beskales, and M. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[12] F. J. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *ICML*, pages 445–453, 1998.

[13] I. Tatarinov. An efficient LFU-like policy for Web caches. Technical report, Computer Science Department, North Dakota State University, 1998.

[14] S. Tatikonda, B. Cambazoglu, and F. Junqueira. Posting list intersection on multicore architectures. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, pages 963–972, 2011.

[15] J. Zhang and T. Suel. Optimized inverted list assignment in distributed search engine architectures. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10. IEEE, 2007.