Shard Ranking and Cutoff Estimation for Topically Partitioned Collections

Anagha Kulkarni* Almer S. Tigelaar‡ Djoerd Hiemstra‡ Jamie Callan*

*Language Technologies Institute, School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue,Pittsburgh, PA, USA 15213

‡Database Group, University of Twente, P.O. Box 217, 7500 AE, Enschede, The Netherlands

{anaghak,callan}@cs.cmu.edu {a.s.tigelaar,hiemstra}@cs.utwente.nl

ABSTRACT

Large document collections can be partitioned into *topical shards* to facilitate distributed search [19]. In a low-resource search environment only a few of the shards can be searched in parallel. Such a search environment faces two intertwined challenges. First, determining *which* shards to consult for a given query: *shard ranking*. Second, *how many* shards to consult from the ranking: *cutoff estimation*. In this paper we present a family of three algorithms that address both of these problems. As a basis we employ a commonly used data structure, the *central sample index* (CSI) [29], to represent the shard contents. Running a query against the CSI yields a *flat* document ranking that each of our algorithms transforms into a tree structure. A bottom up traversal of the tree is used to infer a ranking of shards and also to estimate a stopping point in this ranking that yields cost-effective selective distributed search. As compared to a state-of-the-art shard ranking approach the proposed algorithms provide substantially higher search efficiency while providing comparable search effectiveness.

Categories and Subject Descriptors

H.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Information Search and Retrieval

Keywords

distributed information retrieval, selective search

1. INTRODUCTION

Distributed Information Retrieval (DIR) systems operate by partitioning the document collection into a number of smaller collections (*shards*) which are then searched in parallel [7, 8, 23, 24, 26]. This search strategy has become the de-facto standard in large-scale information retrieval environments [3, 4, 5, 27]. When operationalized on a large computing cluster, distributed search provides excellent gains in query response time [5, 12]. However, in a resource constrained environment where the parallelization opportunities are limited, processing a query against a large document collection is often inefficient. Kulkarni and Callan [19] proposed a solution that is based on dividing the collection into topically clustered partitions: *topical shards*. They demonstrate that such an organization of the collection ensures that the majority of the relevant documents for a given query are concentrated in a small number of shards[1], as opposed to being uniformly distributed across all shards. The skew in the distribution of relevant documents allows the search to be restricted to a few shards without incurring loss in search effectiveness. In this work we study two research problems that arise when selectively searching a partitioned collection: *which* shards to select and *how many* shards to select, specifically in the context of topical shards and low-resource environments.

A shard selection algorithm is responsible for identifying the partitions that should be consulted for a given query. The goal is to maximize the search accuracy by identifying the most *relevant shards* for the query and to simultaneously minimize the cost by searching as few shards as possible. A rich line of work exists for the problem of relevant shard identification and ranking [2, 10, 15, 16, 29, 32]. However, the task of estimating the *minimum number of shards* that should be searched for a given query has not received much attention. Most previous work resorts to using fixed cutoff values on the shard ranking that are query-agnostic [10, 25, 28, 29] or use other query independent criteria, such as, the load on the system to determine the number of shards that would process the query [26].

We show that a fixed rank cutoff either over or underestimates the minimum number of shards for many queries. This either wastes computational resources or returns sub par search results. Neither of these are acceptable choices for most search environments, but it is especially critical for search providers with limited computing resources to use a search strategy that is both efficient and effective.

In this paper we propose a solution that addresses both of the above problems, shard ranking and rank cutoff estimation, simultaneously in order to support cost-effective search. The task of estimating the minimal shard rank for a query is inherently dependent on the predicted ranking of the shards for the query. Solving these problems together

[1]For each query this could be a different set of shards.

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

allows for modeling the intrinsic dynamics between the two components.

All the algorithms studied in this paper assume the availability of a *central sample index* (CSI). This is an inverted index of a small sample of randomly selected documents from each shard. Running a query (q) against this index yields a ranking of the sample documents ($\mathbf{D}^q_{CSI}$). Since each document is associated with a specific shard, a shard ranking can be derived from this. The approach of inferring a shard ranking from $\mathbf{D}^q_{CSI}$ is a well established methodology that has been used in many other algorithms [28, 29, 32]. However, contrary to existing approaches, we derive a tree structure from this *flat* document ranking, using three different approaches: lexical hierarchical clustering, rank inferring, and connected rank inferring. We traverse through this tree to produce a final shard ranking and a minimal cutoff for a given query.

Often the sample used to create the CSI and thus the shard ranking is a very small portion of the complete collection. As such, the shard ranking algorithm is offered only a limited and incomplete view of the collection contents. Short queries can make the task of shard ranking further challenging. The intuition behind overlaying a hierarchy on the CSI document ranking is to capture the relations between the retrieved documents, in terms of their lexical similarities or shard memberships, with the goal of gleaning additional information for shard ranking and rank cutoff estimation.

The details about how we operationalize this intuition are given in Sections 3 through 5. Before that we position our work with respect to the existing work in the following section. The experimental details are shared in Sections 6 and 7. The results and our interpretations of the results are discussed in Section 8. Our conclusions from this investigation are shared in Section 9.

2. RELATED WORK

It has been well established that the problem of *ranking collections of documents* is inherently different from that of *ranking documents* and that it deserves its own solutions. As a result a rich line of research exists for the resource ranking problem which we review in this section. However, the problem of determining the shard cutoff has not received much attention. To the best of our knowledge the cutoff estimator by Thomas and Shokouhi [32] is the only investigation of this problem that is query-specific. In the context of Web search Puppin et al. [26] propose an approach for rank cutoff estimation that makes the number of contacted shards dependent on the current system load.

Several of the early works in shard ranking modeled this task after the well-studied problem of document ranking. The CORI [10] algorithm proposed an adaptation of the INQUERY [9] document ranking algorithm that viewed each shard as one *large document*. Xu and Croft [34] constructed language models (LMs) from the large documents and computed the Kullback-Leibler divergence between the shard LMs and the query LM to rank the shards. GlOSS [15] used term statistics, like the document frequency, in order to estimate the number of relevant documents in each shard for the query. These approaches are examples of a large family of shard ranking techniques that use content statistics of the shards to estimate their relevance for a query.

The other source of information, widely used by many shard ranking algorithms, is the *central sample index* (CSI) data structure. The CSI is created by pooling a sample of documents (S_R) from each shard (R). The ReDDE [29] algorithm runs the user query against the CSI and assumes that the top n retrieved documents are relevant. If n_R is the number of documents in n that are mapped to shard R then a score for each R is computed as $\theta_R = n_R * w_R$, where the shard weight w_R is the ratio of size of the shard ($|R|$) and the size of its sample ($|S_R|$). The shard scores θ_R are then normalized to obtain a valid probability distribution used to rank the shards. CSI creation is query-independent, and is thus performed only once. The CRCS [28] algorithm uses the rank information of the top 50 CSI documents to scale the membership function such that documents at higher ranks contribute more towards resource scores than those at lower ranks. CRCS also scales the accumulated resource scores by their sizes.

SUSHI [32] uses the scores and adjusted ranks of the top 50 CSI documents to fit curves for each resource represented in these documents. Three types of curves, linear, logarithmic, and exponential are tried. The curve with best fit is used to interpolate scores for the top m ranks for each resource. The interpolated points from each resource are merged into a single ranking and scores are aggregated to compute a relevance score for the resources. The number of unique resources present in the top R documents in this ranking is the predicted rank cutoff for the P@R metric. Empirical evaluatation using standard federated search datasets demonstrate that SUSHI predicts lower rank cutoffs on average as compared to a query-agnostic fixed cutoff of 10 and its precision at rank 10 is comparable to that of the baseline approach for many of the datasets.

The following three approaches cannot be categorized into either of the above families. Arguello et al. [2] took a classification based approach that uses CORI and CSI based ranking as classifier features. For each resource a binary classifier is learned that estimates the relevance probability. Document contents, query category and click-through logs are also used to define classifier features. The confidence score assigned for a prediction by each of the classifier is used to rank the shards. The fixed rank cutoffs of 1–5 are evaluated and the results demonstrate that the non-content features together with the conventionally used content features yield a robust ranking approach.

The usefulness of a query's topical category for the task of shard ranking was also noted by Ipeirotis and Gravano [16]. In fact, their approach organizes the shards into a topical hierarchy which is created using samples of shard contents. This hierarchy is static and query independent. The hierarchical shard selection is performed by evaluating the query against all of the shards at a particular level in the hierarchy using one of the *flat* shard ranking algorithms like CORI. The top scored shard at this level is then selected to be searched and the sub-tree rooted at that node is further explored. The experimental results show that this hierarchical shard ranking approach enables higher average precision as compared to the flat shard ranking technique.

Puppin et al. [25] use query logs to organize collections and to perform shard selection. They construct a contingency matrix from the queries observed in the query log and the documents returned for each of them. Co-clustering of this matrix provides *document clusters* and *query clusters*. The former are the partitions of the collection and the latter are used for shard selection. The experimental results show

that a collection of about 6 million documents is partitioned into 18 shards using the proposed approach. One of the shards contains 52% of the collection documents which could not be clustered using the proposed technique. The results show that the proposed approach supports higher precision at early ranks (5, 10 and 20) than CORI.

In the context of geographically partitioned multi-site Web search Cambazoglu et al. [11] present an approach for query forwarding that uses training queries to infer the upper bounds on document scores for a given query at each of the non-local indexes. These bounds are then used to predict if a non-local index would contribute any document to the top k results for the query which in turn determines if the query is forwarded to that non-local index.

Peer-to-peer information retrieval is another line of related research where the task is to find a subset of peers that a query should be forwarded to. However, there is commonly either no central mediator or a light one that only suggests appropriate peers for a given query, but does not participate in the actual retrieval process. Each peer could be said to hold an index shard.

In networks where each peer holds a local index and is connected to a limited number of neighboring peers, the query is typically flooded: the querying peer sends it to its neighbors, and those neighbors send it to theirs, et cetera [1]. Since this does not scale, the forwarding has to be bounded, there is a fixed search horizon that results in a variable cutoff. Whilst this works well for popular queries with prolific results, it fares poorly for long tail queries. Query routing is complicated further by the fact that peers are not naturally topically clustered [31]. Hence, clustering peers by content [14, 20] is a possible solution as it has been shown to make query processing more efficient [6].

A topical cluster of cooperative peers is similar to a topical shard. However, no mechanism exists to dynamically determine the number of peer clusters to contact for a given query in this set-up, for example: Bawa et al. [6] proposed a heuristic cutoff equal to a quarter of the number of topical clusters. Furthermore, clustering mechanism may fail to discover the structure of the document distribution and may not work well for unpopular topics. Klampanos et al. [17] proposed replicating popular documents and using relevance feedback to dynamically alter cluster centroids.

3. TOPICAL SHARDS: RANKING AND CUTOFF ESTIMATION

When large document collections are divided into multiple smaller shards the resulting search environment offers several advantages, such as, distributed and parallel computing, higher fault tolerance, and easier load balancing. Randomly partitioning the collection into shards and then searching all of them in parallel is a well-established search strategy. However, Kulkarni and Callan [19] reason that if the collection is instead divided into topically focused shards, only a few of them need to be searched for each query[2]. They show that this substantially reduces the query processing cost while maintaining search accuracy. Although topical partitions may not support even load-balancing, their lower total search cost makes them an appropriate solution for low-resource search environments that must support large collections.

The topically homogeneous partitions have certain unique properties that can be exploited at query time. The distribution of relevant documents across shards is highly skewed for the majority of queries when shards are topically organized [19][3]. Secondly, many shard ranking algorithms, including ours, rely on a small subset of documents sampled from each shard to sufficiently represent its contents. This objective is harder to achieve when the shards are created using random partitioning. Sampling from topically focused shards is less likely to result in under or over-sampling errors, which in turn improves shard ranking effectiveness.

Motivated by these observations we propose three algorithms that address both shard ranking and cutoff estimation for topically sharded search environments. Later we also test the adaptability of the proposed algorithms by evaluating them on datasets that were not partitioned using the approach proposed by Kulkarni and Callan.

4. Sampling-based Hierarchical Relevance Estimation (SHiRE)

We propose a family of three shard ranking algorithms that share two characteristics: their use of a central sample index (CSI), and their approach to shard ranking that creates a hierarchy from the CSI documents retrieved for the query. What distinguishes these algorithms is the function they use to transform the *flat* CSI document ranking for a query into a tree structure.

In this work we construct the CSI from a small sample of randomly selected documents from each shard. We choose to use the CSI to represent the shard contents for two main reasons. Firstly, it is a time-tested approach that has been demonstrated to be highly effective by several previous studies [2, 29, 32]. Secondly, the CSI typically has a small memory footprint leading to an efficient shard ranking process. Also, note that the CSI creation happens during an offline phase that is not repeated for each query.

Each of the SHiRE algorithms starts with scoring the query against the CSI to obtain a ranking of the sampled documents ($\mathbf{D}^q_{CSI}$). The information contained in this ranking, such as, the rank of each document, the retrieval scores, and the shards represented by each of the retrieved sample documents, is used by each of the algorithms in a unique way to construct a query-specific hierarchy where the retrieved documents are at the leaf nodes. Note that the set of documents organized into a hierarchy for each query is quite small ($\mathbf{D}^q_{CSI} \ll CSI \lll C$, where C is the complete collection). The constructed hierarchy is traversed bottom-up starting at the leaf node that represents the top ranked CSI document for the query, until the root node is reached. Each step up reveals new leaf documents which 'vote' for the shards they represent. However, the votes are exponentially decayed at each level in this traversal. A document d revealed at step U in the traversal contributes a vote toward its parent shard as follows:

$$Vote(d) = V_d * B^{-U} \quad (1)$$

where V_d is either the document score assigned by the retrieval model or is a unit weight, and B is the base of the

[2]Verticals and groups of peers with similar content can also be regarded as instances of topical shards.

[3]For random partitioning the distribution of relevant documents across shards would be close to uniform.

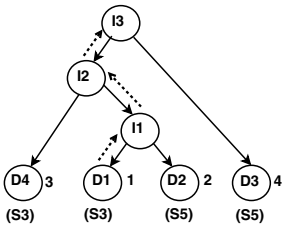

Figure 1: Lexical SHiRE. CSI ranking: $D1, D2, D3, D4$. Dashed arrows represent the path followed by the bottom-up traversal starting at $D1$. Numbers besides the leaf nodes specify the order of revelation of the documents. Parent shards specified in the brackets.

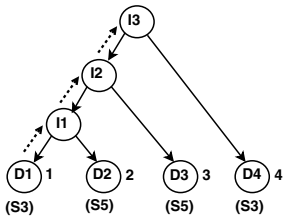

Figure 2: Rank SHiRE. CSI ranking: $D1, D2, D3, D4$. Dashed arrows represent the path followed by the bottom-up traversal starting at $D1$. Numbers besides the leaf nodes specify the order of revelation of the documents. Parent shards specified in the brackets.

exponential function. The resulting votes are accumulated for each of the shards and are interpreted as their estimated relevance scores which are then used to obtain a ranking of the shards for the query.

By anchoring the tree traversal at the top ranked CSI document these algorithms assert that the shard represented by the first document in the CSI ranking is likely to be the most relevant shard for the query. The process of dampening the votes at each step models the intuition that longer path lengths from the anchor node implies less similarity with it which further implies lower likelihood of relevance to the query (*cluster hypothesis* [33]). If a tree is very shallow, few shards would accumulate a zero relevance score, whereas a very deep tree will result in many shards with zero scores.

4.1 Lexical SHiRE (Lex-S)

The *Lex-S* algorithm organizes the CSI documents retrieved for a query into a hierarchy based on their lexical similarities. Such a hierarchy provides equal voting rights to all documents that are similar (attached to the same node in the tree) irrespective of their CSI ranking.

As a first step the Lex-S algorithm represents each of the CSI documents retrieved for the query as a vector of tf-idf weights for the unique terms. The Manhattan distance metric is used to compute pairwise lexical similarities between the document vectors and Ward's method for agglomerative clustering constructs the hierarchy (complexity: $O(n^3)$ where n is the number of CSI documents retrieved for the query, ranging between between 1700–2000 documents in this work). Such a hierarchy for a toy example consisting of four CSI documents retrieved for a query is shown in Figure 1. Dn represents the document at rank n in the CSI results.

The ranking of shards is derived by traversing up this tree, starting from the leaf node for the first document in the CSI ranking ($D1$). For the toy example in Figure 1 the next step would reach the internal node $I1$ and thus reveal the document $D2$. $D4$ would be observed next and $D3$ would be found the last. If V_{D1} is the vote that $D1$ ascribes to its parent shard then $D1$ will contribute: $V_{D1} * B^{-U}$, where U=1 is the number of steps traversed up the hierarchy, and B is the base of the exponential decay function. Similarly, $D2$, $D4$ and $D3$ will contribute: $V_{D4} * B^{-2}$, $V_{D3} * B^{-3}$, and, $V_{D4} * B^{-4}$ respectively, to their parent shards. The shard $S3$ will accumulate votes from documents $D4$ and $D1$, and shard $S5$ will accumulate votes from documents $D2$ and $D3$. The resulting score would be used to rank shards $S3$ and $S5$.

4.2 Rank SHiRE (Rank-S)

The *Rank-S* algorithm uses the rank information of the documents retrieved from CSI for the query to construct the hierarchy. We use a left-branching binary tree to encode the rank information since it provides a convenient structure that retains the CSI ranking during bottom-up traversal. It also facilitates a tree traversal based shard rank inference, which is a common characteristic of this family of algorithms. As shown in the tree representation of the toy example (Figure 2) the first CSI document is at the deepest node in this left-branching binary tree and the height of this tree is one less than the number of documents retrieved from the CSI for the query.

The procedure for inferring a shard ranking from this tree is similar to that used by Lex-S except for a *regularization* enforced on the vote assigned by the single top ranked CSI document. If the shard represented by the top ranked CSI document does not earn any other votes from the subsequent documents at the top m ranks then the vote of the top document is ignored. Recall that by anchoring the traversal at the top ranked document we allow it to assign the biggest vote. The regularization puts a check on the ability of the top ranked document to vote by requiring evidence of support from the subsequent ranks. If the shard represented by the top ranked document is also represented by at least 10% of the ranks in the top 30 CSI documents for the query then its vote is accumulated. We chose these values for the two parameters based on preliminary experiments. A more thorough study of the effects of these parameters is needed.

The simplicity of this algorithm makes it the most efficient algorithm of this family. This approach is most similar in spirit to the CRCS(e) ranking method [28]. However, other CSI based shard ranking methods, like ReDDE, can also be easily transformed to work with a left-branching binary tree of CSI documents where the votes are not decayed during the bottom-up traversal. As such, the computational complexity of Rank-S is very similar to that of ReDDE.

4.3 Connected SHiRE (Conn-S)

The *Conn-S* algorithm uses the *connections* between the retrieved CSI documents, specifically their shard membership, to guide the construction of the hierarchy. The top ranked CSI document is the deepest node in the hierarchy. The next document in the CSI ranking is either attached at the same level as the previous node, or it prompts creation of another level depending upon whether it belongs to the same shard as the previous document. As per the cluster hypothesis, for topical shards, documents from the same

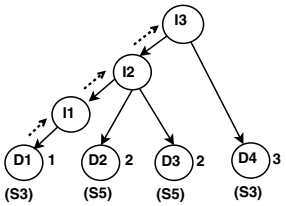

Figure 3: Connected SHiRE. CSI ranking: $D1, D2, D3, D4$. Dashed arrows represent the path followed by the bottom-up traversal starting at $D1$. Numbers besides the leaf nodes specify the order of revelation of the documents. Parent shards specified in the brackets.

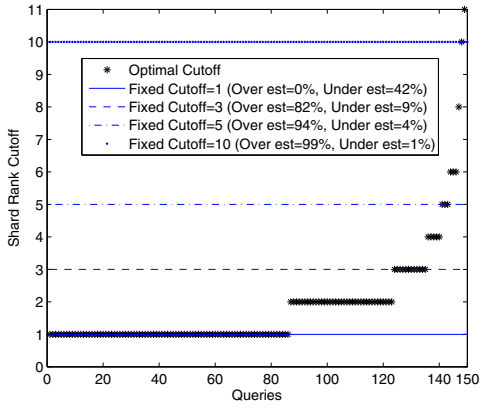

Figure 4: Minimal versus fixed shard rank cutoffs for ReDDE. Dataset: Gov2. Metric: P@10.

shard are likely to be relevant to the same information need. Placing such documents at the same internal node provides them equal voting rights.

In the toy example (Figure 3) document $D1$ belongs to a different shard as $D2$, which is why a new internal node is created as $D2$ is observed. In contrast, $D2$ and $D3$ belong to the same shard, and thus $D3$ attaches to the same node as $D2$. Finally, $D4$ leads to a creation of another internal node because it belongs to a different shard ($S3$) than $D3$ ($S5$). This continues until every document in the CSI ranking is processed. A CSI ranking that does not place any pair of documents from the same shard in consecutive ranks results in a left-branching binary tree using this method. A shard ranking is estimated from the resulting hierarchy using the same approach as that used by Lex-S: bottom-up traversal with exponential decays at each level.

5. SHARD RANK CUTOFF ESTIMATION

Once a ranking of shards is obtained for the query using one of the SHiRE algorithms, the next step is estimating the *minimal rank cutoff* in this ranking. A minimal rank cutoff is the smallest rank in a given shard ranking that provides search accuracy on par with exhaustive search. Notice that the minimal cutoff is specific to a shard ranking. Two different shard rankings for the same query could have different minimal cutoff values. The cutoff also depends on the evaluation metric under consideration.

Figure 4 plots the minimal rank cutoffs for 150 evaluation queries used with one of the datasets in this paper. The minimal rank cutoffs were computed for precision at rank 10 metric (P@10). The straight lines represent the commonly used query-agnostic fixed rank cutoffs. This figure illustrates that the minimal shard rank cutoffs (represented by asterisks) exhibit high variability across queries. A fixed cutoff leads to an under or overestimation error for many queries. A small fixed value, such as 1, would lead to a poor search accuracy for 42% of the queries in this example, while a larger fixed value of 10, would lead to an unwarranted increase in the search cost for 99% of the queries.

The SHiRE algorithms facilitate estimation of a query-specific minimal cutoff. Recall that the bottom-up traversal is accompanied by exponential decay of votes for each of the algorithms. This has the effect of driving the votes to zero as the traversal progresses which in turn drives the relevance scores, which are simply the accumulated votes, to zero for shards that are represented only higher up in the tree. Our hypothesis is that in the resulting shard ranking the point at which a shard's estimated relevance score converges to zero is often close to the minimal cutoff for the query. We use this rank cutoff estimator for each of the three SHiRE algorithms and interpret a shard's relevance score of 0.0001 as having converged to zero.

Table 1: Datasets

Dataset (#Shards)	Size (GB)	#Doc (K)	#Voc (M)	Avg Doc Len
CW09-CatB (100)	1500	50220	96	918
Gov2 (50)	400	25205	39	918
TREC123-BySrc (100)	3.2	1078	0.9	420
TREC4-Kmeans (100)	2.0	567	0.6	480

Table 2: Query Sets

Dataset (#Queries)	Qry Set	Avg Qry Len	Avg #Rel Docs / Qry	Num Rel Grades
CW09-CatB (98)	0910:1-100	2.1	80 (± 49)	5
Gov2 (149)	701-850	3.1	179 (± 149)	3
TREC123-BySrc (50)	51-100	3.4	546 (± 375)	2
TREC4-Kmeans (50)	201-250	9.1	130 (± 115)	2

6. DATASETS

For a thorough evaluation of our algorithms we employ four datasets that provide a range of sizes and characteristics. The first two datasets described below (CW09-CatB and Gov2) are the main datasets that we use throughout the evaluation section. We use the other two datasets (TREC4-Kmeans and TREC123-BySrc) as supplementary datasets for their historic value and to test the adaptability of our approaches to different partitioning environments.

The two main datasets were partitioned into topical shards using a sample-based clustering technique proposed by Kulkarni and Callan [19]. This technique learns the *shard definitions* from a small randomly sampled subset of the collection. The remaining documents are then projected onto the topical spaces using the learned definitions.

The CW09-CatB dataset is a subset of the ClueWeb09 dataset. It consists of the first 50 million English Web pages of ClueWeb09. We divided this dataset into 100 topical shards obtaining an average shard size of 0.5 million, as

specified by Kulkarni and Callan [19]. The Gov2 TREC corpus [13] consists of 25 million documents from the US government domains, such as .gov and .us, and also from certain government related websites. This dataset was partitioned into 50 shards using the above topical shard creation technique.

Although they are now considered small and outdated, TREC4-Kmeans and TREC123-BySrc are two of the most widely used datasets in distributed information retrieval research. TREC4-Kmeans was created by clustering the documents from the Text Research Collection, Volumes 2 and 3[4] into 100 clusters [34]. TREC123-BySrc dataset was created by dividing the Text Research Collection, Volumes 1, 2 and 3, into 100 partitions based on the source of the documents.

The summary statistics of these datasets are given in Table 1. The evaluation queries that were used with these datasets are summarized in Table 2. For CW09-CatB there are two queries and for Gov2 there is one query for which there are no relevant search results. These were not included conforming to the official TREC evaluations. Note that the CW09-CatB and Gov2 queries were judged on a 5-point and 3-point relevance scales, respectively, while the older datasets provide only binary relevance judgments.

7. EXPERIMENTAL SETUP

For efficient and convenient access to the document collections, each dataset was converted to an Indri[5] index. The documents were pre-processed before indexing to remove stopwords and reduce morphological variants using the Krovetz stemmer [18].

A central sample index (CSI) was created for each dataset using simple random sampling. From each shard 4% of the documents were sampled for the CSI. The choice of the sample size was guided by preliminary experiments that were performed to estimate the size of a small but effective sample. For each dataset the same CSI was used by all the shard ranking methods. Global statistics were assumed to be available to all the shards. As a result, the scores of the documents retrieved from different shards are comparable and the merging of documents is straightforward. A language modeling and inference network based retrieval model, Indri [21], was used for all the document retrieval tasks. The full-dependence model query representation [22], that encodes term co-occurrence information, was used for all the experiments.

For the comparative analysis of the different shard ranking approaches we use the following standard evaluation metrics: precision at rank 10, rank 30 (P@{10,30}), normalized discounted cumulative gain at rank 10 (ndcg@10), and the mean-average-precision (MAP) metric to measure the average search accuracy over all the queries. In order to compare the search efficiency enabled by each of the approaches we define a *cost* metric. The total amount of work required to process a query against a collection is typically proportional to the number of documents that *match* the query[6]. The volume of index data transferred and processed for a query (the dominant cost) as well as the computational cost of the query, both, are functions of the number of matched documents. We define the cost metric as follows:

$$cost = \frac{\sum_{q=1}^{|Q|} \sum_{t=1}^{T_q} |D_{S_t}|}{|Q|} \quad (2)$$

where Q is the set of evaluation queries, T_q is the number of top shards searched for the query q, and D_{S_t} is the set of matched documents in shard S at rank t in the predicted shard ranking for query q. Note that T_q would be a fixed value for each query in case of ReDDE. For the SHiRE algorithms T_q would be a query-specific value provided by the rank cutoff estimator.

The choice of this cost metric is especially well-suited for low-resource search environments, where the operational requirements place an upper bound on the amount of work permitted per query. Similar cost metrics have been used in other research by Moffat et al. [23] and Strohman et al. [30]. Most real-world search systems use query optimization techniques to improve query processing efficiency. The actual average search cost would typically be lower than the one computed above.

8. RESULTS AND DISCUSSION

Tables 3 and 4 report the search accuracy and cost results for the Gov2 and CW09-CatB datasets. The SHiRE algorithms with dynamic shard rank cutoff estimation are compared with two previously proposed shard rankers: ReDDE and SUSHI. Several different parameterizations were tested and the setting that provided the best trade-off in terms of search accuracy and processing cost was chosen for each algorithm. For ReDDE the parameter under consideration is the fixed shard rank cutoff (T) and for the SHiRE algorithms it is the base for the exponential decay function (B). The chosen values for these parameters are provided in the first column of the tables. The sensitivity of SHiRE algorithms to different parameterizations is analyzed later in Section 8.1. The differences in the search accuracies of ReDDE and the proposed approaches were tested for statistical significance using the paired T-test and were found to be not significant ($p < 0.01$). The last column in both tables compares each algorithm's query processing efficiency with that of ReDDE.

We observe that all the three SHiRE algorithms support a more efficient search than ReDDE. The average number of documents processed per query is smaller for the SHiRE algorithms while the search effectiveness is on par with that of ReDDE. The Rank-S algorithm cuts the search cost by 27% for Gov2 and by nearly half for CW09-CatB. The savings in cost offered by Lex-S and Conn-S are bigger for the larger collection. This is a desirable property for low-resource search environments. Providing cost-effective search, even when working with large collections, is one of the challenges in such search environments.

The shard rank cutoff estimator used by SUSHI optimizes its prediction for a particular precision metric (Section 2). The P@10 and P@30 values reported for SUSHI are thus from separate runs and the remaining values are an average over those two runs. For the Gov2 dataset SUSHI is as efficient as the Lex-S algorithm. However, the corresponding search accuracy for SUSHI is substantially lower than ReDDE. For the CW09-CatB dataset, although the search accuracy results for SUSHI are comparable to other

[4]http://trec.nist.gov/data/docs_eng.html
[5]http://www.lemurproject.org/indri/
[6]Every document that contains at least one of the query terms is considered as a matched document for the query.

Table 3: Search accuracy and cost results for Gov2. Exhaustive search performance: P@10=0.58, P@30=0.52, MAP=0.32, ndcg@10=0.49, Cost=3.62M ▼ denotes significantly worse value than that with ReDDE ($p < 0.01$).

	P@10	P@30	MAP	ndcg@10	Cost (million)	$\text{Cost}_{MtdX} - \text{Cost}_{ReDDE}$
ReDDE (T=3)	0.57	0.51	0.29	0.48	0.83	-
SUSHI	▼0.51	▼0.41	▼0.23	▼0.41	0.77	-7%
Lex-S (B=10)	0.56	0.49	0.27	0.46	0.78	-7%
Rank-S (B=10)	0.58	0.52	0.29	0.48	0.61	-27%
Conn-S (B=10)	0.58	0.52	0.31	0.48	0.81	-2%

Table 4: Search accuracy and cost results for CW09-CatB. Exhaustive search performance: P@10=0.27, P@30=0.26, MAP=0.18, ndcg@10=0.21, Cost=5.37M.

	P@10	P@30	MAP	ndcg@10	Cost (million)	$\text{Cost}_{MtdX} - \text{Cost}_{ReDDE}$
ReDDE (T=3)	0.29	0.28	0.17	0.21	0.47	-
SUSHI	0.29	0.26	0.16	0.21	0.54	15%
Lex-S (B=50)	0.29	0.26	0.15	0.22	0.39	-17%
Rank-S (B=50)	0.30	0.26	0.16	0.21	0.25	-47%
Conn-S (B=50)	0.28	0.26	0.16	0.20	0.31	-34%

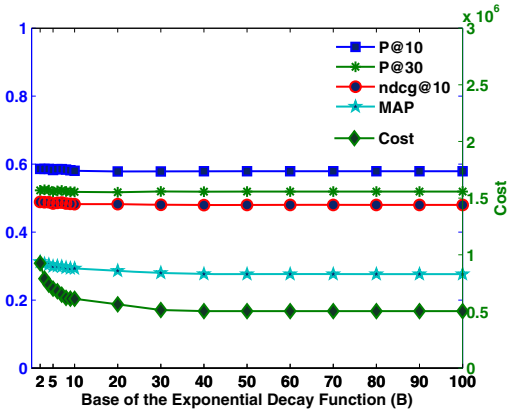

Figure 5: Sensitivity of Rank-S algorithm to parameter B. Dataset: Gov2

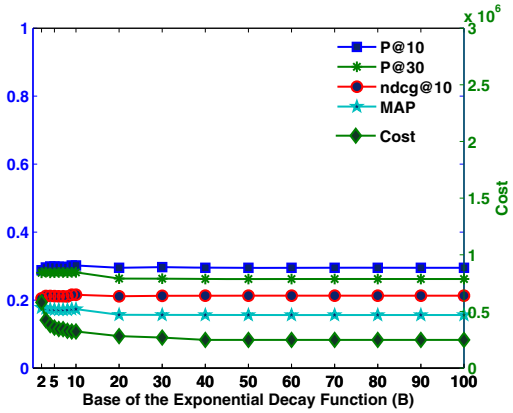

Figure 6: Sensitivity of Rank-S algorithm to parameter B. Dataset: CW09-CatB.

approaches the corresponding cost is much higher than all the other approaches. Overall SUSHI struggles to provide a cost-effective search setup and as such is a weaker baseline than ReDDE. In the remainder of the paper we compare the SHiRE algorithms only with the ReDDE algorithm.

We also note that almost all the shard rankers support efficient search that is just as effective as the exhaustive search. This is especially true for precision at early ranks. For both datasets, recall that the shards were created by partitioning the collection's documents into topical shards (Section 6). We see that all the ranking approaches benefit from the characteristic property of topical shards: a skewed distribution of relevant documents across shards (Section 3). These result trends are in agreement with observations made by Kulkarni and Callan [19].

Notice that even for ndcg@10, a metric that is sensitive to both the position and the relevance level of the documents in the results list, nearly all of the rankers are able to provide comparable values to those with exhaustive search. This is especially remarkable for the CW09-CatB dataset which offers 5 levels of graded relevance.

When comparing the SHiRE algorithms to each other we see that the Rank-S approach provides the largest reduction in cost for both datasets. The computational complexity of the Lex-S algorithm is much higher than the other two SHiRE algorithms. The transformation of the CSI results into a hierarchy requires agglomerative clustering of the results in case of Lex-S, whereas this transformation is trivial for Rank-S and Conn-S. Based on the above observations we recommend the SHiRE algorithms (especially Rank-S) over ReDDE for topically partitioned collections.

8.1 Sensitivity of SHiRE algorithms to parameter B

We experimented with a range of base values (B) for the exponential decay function (Equation 1) in order to analyze its influence on the performance of the SHiRE algorithms. Figures 5 and 6 present the results for the Rank-S algorithm with both datasets. The results for each of the precision metrics are fairly stable over a range of B values. We see more variation in the values for the cost metric. A small base value allows for more search budget (in terms of shard rank cutoff). As a result, the corresponding search cost increases as the base value decreases.

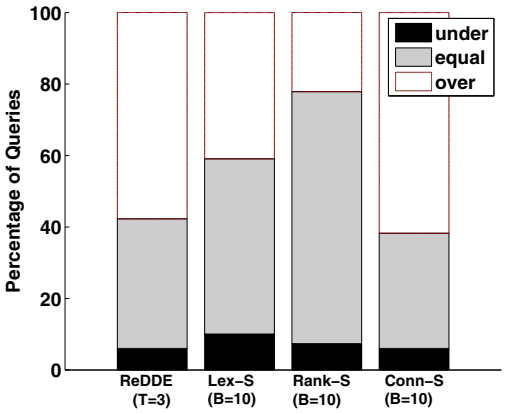

Figure 7: Shard rank cutoff estimation errors. Dataset: Gov2. Metric: P@10.

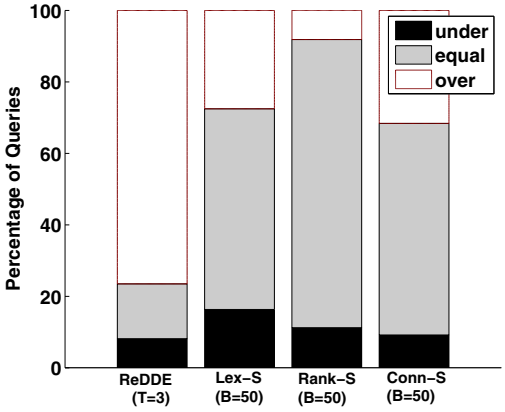

Figure 8: Shard rank cutoff estimation errors. Dataset: CW09-CatB. Metric: P@10.

We see similar trends for the Conn-S and Lex-S algorithms. However, for the latter the cost is much higher for smaller base values. Often the hierarchy learned from the CSI results of a query for the Lex-S algorithm is more flat than it is for the Rank-S and Conn-S algorithms. As a result, the votes accumulated at each level in the tree, decay at a much slower rate for the Lex-S algorithm. This results in a higher shard rank cutoff estimates and ultimately to higher search costs. Overall, these results demonstrate that the SHiRE algorithms are not highly sensitive to the parameter B and that we see consistent trends that are intuitive.

8.2 Shard Rank Cutoff Estimation

In this section we analyze the effectiveness of the SHiRE algorithms at predicting the minimal shard rank cutoff. We categorize the cutoff estimates into: under, equal and over estimates, based on comparison with the minimal rank cutoff. The equal category has a tolerance of ±1. An estimate that is off by +1 or -1 from the minimal value would be counted as an accurate estimate for this analysis. Recall that the minimal cutoff value for a query is specific to a metric. In these results the metric is P@10. We see similar trends for the other metrics.

Figures 7 and 8 plot the distribution of the estimation errors for the Gov2 and CW09-CatB datasets, respectively.

Table 5: Confusion matrix for shard rank cutoff estimation for Rank-S (B=10). Dataset: Gov2. Metric: P@10. Queries: 149. Cutoff averages: optimal=1.8 (±1.9), predicted=2.3 (±1.0).

		Predicted					
		1	2	3	4	5	>5
Optimal	1	34	37	26	5	2	0
	2	0	7	12	0	0	0
	3	0	4	5	2	0	0
	4	0	2	1	3	0	0
	5	1	1	0	0	0	0
	>5	1	2	3	1	0	0

Table 6: Confusion matrix for shard rank cutoff estimation for Rank-S (B=50). Dataset: CW09-CatB. Metric: P@10. Queries: 98. Cutoff averages: optimal=3.1 (±9.3), predicted=1.8 (±0.6).

		Predicted					
		1	2	3	4	5	>5
Optimal	1	28	44	8	0	0	0
	2	0	2	1	0	0	0
	3	2	4	0	0	0	0
	4	0	3	0	0	0	0
	5	0	1	0	0	0	0
	>5	0	5	0	0	0	0

We see that the over estimation errors are more frequent than the under estimation errors for nearly all the predictors. A fixed rank cutoff leads to over-estimation for the majority of the queries in case of ReDDE. This explains the large search costs associated with this search algorithm. All of the SHiRE algorithms lead to fewer prediction errors than ReDDE. However, Rank-S is most successful at predicting the minimal cutoff values (accuracy 71% and 81%, respectively for Gov2 and CW09-CatB) which explains its cost-effective search performance in Tables 3 and 4.

The above analysis provides a high-level validation of the minimal cutoff estimator's efficacy. For a more thorough understanding we study the *magnitude* of the cutoff prediction errors for the Rank-S algorithm. The confusion matrices in Tables 5 and 6 present these values for the Gov2 and CW09-CatB datasets, respectively. We see that the distribution of minimal rank cutoffs is highly skewed toward low cutoff values, specifically 1, for both datasets. This is mainly caused by the topical partitioning of the collection.

If we tolerate an estimation error of magnitude ±1 then the Rank-S estimator is accurate for 71% of the queries for the Gov2 dataset and for 81% of the queries for CW09-CatB. For ReDDE with a fixed cutoff (T) of 3 these numbers are 36% and 15% for Gov2 and CW09-CatB, respectively. The query-specific cutoff estimates by Rank-S are at least twice as accurate as the query-agnostic fixed cutoff.

Tables 5 and 6 show that for many queries the magnitude of the estimation error is not large for either datasets. However, for Gov2 we see a higher rate of over-estimation errors (22%), especially for the minimal cutoff of 1, than for CW09-CatB (8%). On the other hand, under-estimation errors occur for a larger percentage of queries (11%) for CW09-CatB than for Gov2 (7%). The prediction accuracy is especially low for the higher values of minimal cutoffs. Also, the magnitude of these errors is relatively higher. These errors primarily stem from the bias of the Rank-S algorithm

Table 7: Search accuracy and cost results for TREC123-BySrc. For ReDDE the top 25 shards (T) were searched for each query. For SHiRE algorithms the exponential base value of 3 (B) was used. Exhaustive search performance: P@10=0.48, P@30=0.47, MAP=0.21, Cost=0.15M, ▼ denotes significantly worse precision than ReDDE ($p < 0.01$)

	P@10	P@30	MAP	Cost (million)	$\text{Cost}_{MtdX} - \text{Cost}_{ReDDE}$ (%)
ReDDE	0.48	0.46	0.14	0.06	-
Lex-S	0.52	0.46	0.13	0.05	-24%
Rank-S	0.46	▼0.40	▼0.07	0.02	-73%
Conn-S	0.50	0.42	▼0.08	0.02	-71%

Table 8: Search accuracy and cost results for TREC4-Kmeans. For ReDDE the top 20 shards (T) were searched for each query. For SHiRE algorithms the exponential base value of 3 (B) was used. Exhaustive search performance: P@10=0.46, P@30=0.35, MAP=0.21, Cost=0.19M.

	P@10	P@30	MAP	Cost (million)	$\text{Cost}_{MtdX} - \text{Cost}_{ReDDE}$ (%)
ReDDE	0.45	0.34	0.19	0.09	-
Lex-S	0.45	0.33	0.15	0.05	-46%
Rank-S	0.40	0.32	0.16	0.02	-80%
Conn-S	0.41	0.33	0.16	0.02	-77%

toward smaller cutoff predictions, especially when parameterized with higher B values.

Overall, this query-specific rank cutoff estimator takes a step in the right direction and offers substantial improvements over the query-agnostic fixed cutoff approach. However, the above analysis also reveals the areas in which the estimator could be improved.

8.3 Additional Datasets

In this section we test the adaptability of the proposed shard rankers. We experiment with datasets that have not been topically partitioned using the technique proposed by Kulkarni and Callan [19]. We use the TREC123-BySrc and TREC4-Kmeans datasets described in Section 6. These datasets differ from those used in the previous sections on several dimensions, such as: size, average number of relevant documents per query, and topical shard homogeneity. The results for both datasets are given in Tables 7 and 8. As with the first two datasets, the differences in precision values of ReDDE and the SHiRE algorithms were tested for significance using the paired T-test ($p < 0.01$).

The prominent deviation from the trends seen until now is the low performance of the Rank-S algorithm. Although it provides substantial savings in search cost for both datasets, the corresponding search effectiveness is not comparable to the baseline performance. Instead, the Lex-S algorithm supports a more cost-effective partial search for these datasets. This trend reveals one of the weaknesses of Rank-S: its strong bias for predicting small rank cutoffs. At the same time it brings to light the strength of the Lex-S algorithm: its ability to support a more varied cutoff prediction range.

From these analysis we can conclude that the family of SHiRE algorithms together provide a search approach that has reasonable adaptability and is cost-effective for different search needs.

9. CONCLUSIONS

We presented three SHiRE shard ranking algorithms: Lex-S, Rank-S and Conn-S, that transform a flat CSI document ranking into a tree structure in order to encode additional information about the documents and the shards. The constructed hierarchy is also employed for estimating a query-specific minimal shard rank cutoff. The presented algorithms are one of the few approaches to dynamically predict query-specific shard rank cutoffs. The joint formulation of the two inter-dependent problems of shard ranking and cutoff estimation is an additional contribution of this work.

We tested the proposed algorithms on two large datasets that were both partitioned into topical shards. The search accuracy of the SHiRE algorithms is on par with a strong baseline and also with exhaustive search, while the search cost is substantially lower than both. The Rank-S algorithm is the most efficient search method for large topically partitioned collections. It reduced the search cost by a quarter for one of the datasets and by nearly a half for the larger dataset. Rank-S also supports query-specific minimal cutoff estimation that is at least twice as accurate as the best fixed cutoff value for topical shards.

Experiments with two supplementary datasets demonstrated that the conservative nature of the Rank-S cutoff estimator that enables very efficient search can be a detriment for smaller and topically less focused collections. However, Lex-S provides an attractive solution that is efficient and yet effective as compared to the baseline. Overall, the family of SHiRE algorithms offers cost-effective solutions for different search requirements.

10. ACKNOWLEDGMENTS

This work was in part supported by the National Science Foundation (NSF) grant IIS-0916553 and by the Netherlands Organisation for Scientific Research (NWO) under project 639.022.809. Any opinions, findings, conclusions and recommendations expressed in this paper are the authors' and do not necessarily reflect those of the sponsors.

11. REFERENCES

[1] K. Aberer and M. Hauswirth. An overview on peer-to-peer information systems. In *Proceedings of the Workshop on Distributed Data and Structures*, 2002.

[2] J. Arguello, J. Callan, and F. Diaz. Classification based resource selection. In *Proceeding of the ACM Conference on Information and Knowledge Management*, pages 1277–1286, 2009.

[3] R. Baeza-Yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed Web retrieval. In *The International Conference on Data Engineering*, pages 6–20, 2007.

[4] R. Baeza-Yates, V. Murdock, and C. Hauff. Efficiency trade-offs in two-tier Web search systems. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 163–170, Boston, MA, USA, 2009.

[5] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The Google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[6] M. Bawa, G. S. Manku, and P. Raghavan. SETS: Search enhanced by topic segmentation. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 306–313, July 2003.

[7] B. Cahoon, K. S. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Trans. Inf. Syst.*, 18(1):1–43, Jan. 2000.

[8] J. Callan. Distributed information retrieval. In *Advances in Information Retrieval*, pages 127–150. Kluwer Academic Publishers, 2000.

[9] J. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the International Conference on Database and Expert Systems Applications*, pages 78–83, 1992.

[10] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, NY, USA, 1995.

[11] B. B. Cambazoglu, E. Varol, E. Kayaaslan, C. Aykanat, and R. Baeza-Yates. Query forwarding in geographically distributed search engines. In *Proceedings of the ACM SIGIR conference on Research and development in information retrieval*, pages 90–97, NY, USA, 2010.

[12] A. Chowdhury and G. Pass. Operational requirements for scalable search systems. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 435–442, USA, 2003.

[13] C. Clarke, N. Craswell, and I. Soboroff. Overview of the TREC 2004 Terabyte track. In *Proceedings of the 2004 Text Retrieval Conference*, 2004.

[14] A. Crespo and H. Garcia-Molina. Semantic overlay networks for P2P systems. In *Proceedings of the Agents and Peer-to-Peer Computing*, pages 1–13, Heidelberg, DE, July 2004. Springer.

[15] L. Gravano, H. García-Molina, and A. Tomasic. GlOSS: Text-source discovery over the internet. *ACM Transactions on Database Systems*, 24:229–264, 1999.

[16] P. G. Ipeirotis and L. Gravano. Distributed search over the hidden Web: Hierarchical database sampling and selection. In *Proceedings of Conference on Very Large Data Bases*, pages 394–405, 2002.

[17] I. Klampanos and J. M. Jose. An evaluation of a cluster-based architecture for peer-to-peer information retrieval. In *Proceedings of Database and Expert Systems Applications*, pages 380–391, Sept. 2007.

[18] R. Krovetz. Viewing morphology as an inference process. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–202, NY, USA, 1993.

[19] A. Kulkarni and J. Callan. Document allocation policies for selective searching of distributed indexes. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 449–458, 2010.

[20] J. Lu and J. Callan. Content-based peer-to-peer network overlay for full-text federated search. In *Proceedings of RIAO, Conference Adaptivity, Personalization and Fusion of Heterogeneous Information*, 2007.

[21] D. Metzler and W. B. Croft. Combining the language model and inference network approaches to retrieval. *Information Processing and Management*, 40(5):735–750, 2004.

[22] D. Metzler and W. B. Croft. A markov random field model for term dependencies. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 472–479, NY, USA, 2005.

[23] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10(3):205–231, 2007.

[24] S. Orlando, R. Perego, and F. Silvestri. Design of a parallel and distributed web search engine. In *Proceedings of Parallel Computing Conference*, pages 197–204. College Press, 2001.

[25] D. Puppin, F. Silvestri, and D. Laforenza. Query-driven document partitioning and collection selection. In *Proceedings of the Conference on Scalable Information Systems*, page 34, NY, USA, 2006.

[26] D. Puppin, F. Silvestri, R. Perego, and R. Baeza-Yates. Tuning the capacity of search engines: Load-driven routing and incremental caching to reduce and balance the load. *ACM Transactions on Information Systems*, 28(2):5:1–5:36, June 2010.

[27] K. M. Risvik, Y. Aasheim, and M. Lidal. Multi-tier architecture for Web search engines. *Web Congress, Latin American*, 0:132, 2003.

[28] M. Shokouhi. Central-rank-based collection selection in uncooperative distributed information retrieval. In *The European Conference on Information Retrieval*, Rome, Italy, 2007.

[29] L. Si and J. Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 298–305, NY, USA, 2003.

[30] T. Strohman, H. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–225, NY, USA, 2005.

[31] T. Suel, C. Mathur, J.-w. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. ODISSEA: A peer-to-peer architecture for scalable Web search and information retrieval. In *Proceedings of the Workshop on the Web and Databases*, pages 67–72, June 2003.

[32] P. Thomas and M. Shokouhi. SUSHI: Scoring scaled samples for server selection. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 419–426, NY, USA, 2009.

[33] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

[34] J. Xu and W. B. Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 254–261, NY, USA, 1999.