Query-Driven Document Partitioning and Collection Selection

(Invited Paper)

Diego Puppin, Fabrizio Silvestri, and Domenico Laforenza
Istituto di Scienza e Tecnologie dell'Informazione (A. Faedo)
Consiglio Nazionale delle Ricerche
Pisa, Italy
Email: {diego.puppin,fabrizio.silvestri,domenico.laforenza}@isti.cnr.it

***Abstract*— We present a novel strategy to partition a document collection onto several servers and to perform effective collection selection. The method is based on the analysis of query logs. We proposed a novel document representation called *query-vectors* model. Each document is represented as a list recording the queries for which the document itself is a match, along with their ranks. To both partition the collection and build the collection selection function, we co-cluster queries and documents. The document clusters are then assigned to the underlying IR servers, while the query clusters represent queries that return similar results, and are used for collection selection. We show that this document partition strategy greatly boosts the performance of standard collection selection algorithms, including CORI, w.r.t. a round-robin assignment. Secondly, we show that performing collection selection by matching the query to the existing query clusters and successively choosing only one server, we reach an average precision-at-5 up to 1.74 and we constantly improve CORI precision of a factor between 11% and 15%. As a side result we show a way to select rarely asked-for documents. Separating these documents from the rest of the collection allows the indexer to produce a more compact index containing only relevant documents that are likely to be requested in the future. In our tests, around 52% of the documents (3,128,366) are not returned among the first 100 top-ranked results of any query.**

I. Introduction

Millions of new Web pages are created every month. The Web is getting richer and richer and is storing a vast part of the information available worldwide: it is becoming, to many users in the world, a tool for augmenting their knowledge, supporting their theses, comparing their ideas with reputable sources. Differently from libraries, though, the Web's growth and structure is not under the control of a central librarian: contents and links are added and changed without any supervision.

This is why web search engines are among the most used applications of modern information technology. The size of the data available to search engines and information retrieval (IR) systems in general is growing exponentially: very sophisticated techniques are needed to implement efficient search strategies for very large data-bases. Parallel and distributed information retrieval systems are a way to tackle this problem.

A parallel information retrieval system is usually deployed on large clusters of servers running multiple *IR core* modules, each of which is responsible for searching a partition of the whole index. When each sub-index is relative to a disjoint sub-collection of documents, we have a *document-partitioned* index organization, while when the whole index is split, so that different partitions refer to a subset of the distinct terms contained in all the documents, we have a *term-partitioned*, index organization. Each organization of the index requires a specific process to evaluate queries, and involves different costs for computing and I/O, and network traffic patterns.

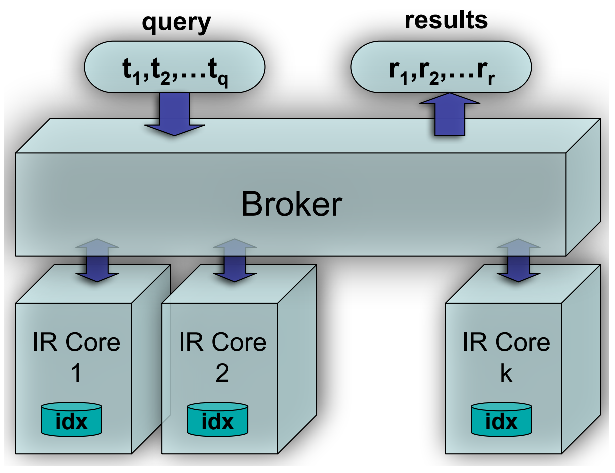

Fig. 1. Organization of a parallel information retrieval system.

In both cases, in front of the cluster, we have an additional machine hosting a *broker*, which has the task of scheduling the queries to the various servers, and collecting the results returned back. The broker then merges and orders the results received on the basis of their relevance, produces the ranked list of matching documents, and it builds the results page containing URL, titles, snippets, and so on. This page is finally returned to the user.

Figure 1 shows the logical organization of a parallel IR system, where k is the number of servers hosting IR Core modules, q the number of terms in a given query, and r the number of ranked results returned for the query.

Document partitioning is the strategy usually chosen by the most popular web search engines [3]. In the document-partitioned organization, the broker may choose among two possible strategies for scheduling a query. A naïve, yet very common, way of scheduling queries is to broadcast each of them to all the underlying IR cores. This method has the advantage of enabling a perfect load balancing among all of the servers. On the other hand it has the major drawback of exploiting all the servers for each query submitted. The other possible way of scheduling is to choose, for each query, the most authoritative server(s). By doing, we reduce the number of IR cores queried. Relevance of each server to a given query is computed by means of a collection selection function that is built, usually, upon statistics computed over each sub-collection.

In this work, we present a novel strategy to perform document partitioning and collection selection. We use the query log to drive the assignment of documents, with the goal of putting together the documents that answer a given query. We do this, first, by representing each document as a *query-vector*, i.e. a (sparse) vector listing the queries to which it answers, weighted with the search score and, second, by performing co-clustering [10] on the query-document contingency matrix. The resulting document clusters are used to partition the documents onto IR cores, while the query clusters are used to perform our collection selection strategy.

Main contributions of this paper are:

1) a new representation of documents as *query-vectors*;
2) a novel representation of collections based on the co-clustering results;
3) a new strategy for document partitioning and collection selection, that proved to be more effective than CORI [7];
4) a way to identify rarely asked-for documents: we showed that about half of the documents of our collection are not returned as top-scoring results of any query.

The last finding means that these documents could be removed from the collection without a significant loss of precision.

This contribution is structured as follows. In the next section, we discuss some related work. Then, we describe the way our data are modeled and the way we applied co-clustering. In Section IV, we show the architecture of our system and how we perform collection selection. Section V shows the experimental results. Finally, we conclude and present our future work.

II. Related Work

Document partitioning (DP) has been shown by many researchers to be the best choice among parallelization scheme. It offers, in fact, the best tradeoff between load balancing and load distribution.

There are a number of papers evaluating DP parallel IR systems; see for instance [1], [4], [5], [32], [34]

All of the above mentioned studies adopt a common architecture for parallel IRSs. It follows the *master/worker* model where workers are the actual search modules which receive queries from and return results to the master that is also known as the *query broker* (QB).

DP has the primary goal of load balancing [2]. Documents are split up randomly. Each query is broadcasted to all the servers, and results from each of them are merged. Clearly, with a random distribution, the average load is smoothly distributed but it is also generally high.

Term partitioning is another technique [2]. The inverted lists describing which documents contain a given term, are divided among a bunch of servers. For each query, the system forwards the query to the servers holding the list related to the query terms. Common terms (or commonly asked-for terms) drive the load.

In all the above cases index is generally randomly partitioned without following any particular ordering.

A very different approach is followed by pSearch [40] and SSW [29]. pSearch performs an initial LSI transformation that is able to identify *concepts* out of the document base. LSI (Latent Semantic Indexing) works by performing a singular value decomposition (SVD) of the document-term matrix, which projects the term vector onto a lower-dimensional semantic space. Then, the projected vector are mapped onto a *Content Addressable Network* CAN [38]. Search and insertion is done using the CAN space. In other words, when a query is submitted, it is projected using the SVD to the CAN, and then neighbors are responsible for answering with the documents they hold.

SSW [29] also uses LSI to reduce the dimensionality of data, but then uses a small-world network rather than a CAN. Again, a query is answered by the neighbors of the node responsible for the query vector, once mapped on the LSI-transformed space.

Today, it is commonly held that realistic parallel IR systems will have to manage distinct indexes. In our opinion, a possible way to ensure timely and economic retrieval is designing a QB module so that it will forward a given query only to workers managing documents related to the query topic. In other words, we are speaking about adopting *collection selection* techniques aimed at reducing a query search space by querying only a subset of the entire group of Workers available. Nevertheless, particular attention should be paid in using this technique. In fact, it could result in a loss of relevant documents thus obtaining degradation in the effectiveness figure.

In the last ten years a large number of research works dealt with the collection selection problem [6]–[9], [11], [12], [14]–[27], [30], [31], [33], [35]–[37], [39], [41], [43]–[45].

The most common approaches to distributed retrieval exploit a number of heterogeneous collections grouped by source and time period. A *collection selection index (CSI)* summarizing each collection as a whole, is used to decide which collections are most likely to contain relevant documents for the query. Document retrieval will actually only take place at these collections. In [23] and [9] several selection methods have been compared. The authors showed that the naïve method of using only a collection selection index lacks in effectiveness. Many proposals tried to improve both the effectiveness and the efficiency of the previous schema.

Moffat *et al.* [33] use a centralized index on blocks of

B documents. For example, each block might be obtained by concatenating documents. A query first retrieves block identifiers from the centralized index, then searches the highly ranked blocks to retrieve single documents. This approach works well for small collections, but causes a significant decrease in precision and recall when large collections have to be searched.

In [20]–[22], [41], H. Garcia–Molina *et. al.* propose GlOSS, a broker for a distributed IR system based on the boolean IR model. It uses statistics over the collections to choose the ones which better fits the user's requests. The authors of GlOSS make the assumption of independence among terms in documents so, for example, if term A occurs f_A times and the term B occurs f_B times in a collection with D documents, than they estimated that $\frac{f_A}{D} \cdot \frac{f_B}{D} \cdot D$ documents contain both A and B. In [19] the authors of GlOSS generalize their ideas to vector space IR systems (gGlOSS), and propose a new kind of server hGlOSS that collects information for several GlOSS servers and select the best GlOSS server for a given query.

In [7], the authors compare the retrieval effectiveness of searching a set of distributed collections with that of searching a centralized one. The system they use to rank collections is an inference network in which leaves represent document collections, and representation nodes represent the terms that occur in the collection. The probabilities that flow along the arcs can be based upon statistics that are analogous to tf and idf in classical document retrieval: document frequency df (the number of documents containing the term) and inverse collection frequency icf (the number of collections containing the term). They call this type of inference network a *collection retrieval inference network*, or *CORI* for short. They found no significant differences in retrieval performance between distributed and centralized searching when about half of the collections on average were searched for a query. Since the total number of collections was small (approximately 17), and the percentage of collections searched was high, their results may not reflect the true retrieval performance in a realistic environment.

CVV (*Cue-Validity Variance*) is proposed in [45]. It is a a new collection relevance measure, based on the concept of *cue-validity* of a term in a collection: it evaluates the degree to which terms discriminate documents in the collection. The paper shows that effectiveness ratio decreases as very similar documents are stored within the same collection.

In [43] the authors evaluate the retrieval effectiveness of distributed information retrieval systems in realistic environments. They propose two techniques to address the problem. One is to use phrase information in the collection selection index and the other is query expansion. In [11] Dolin *et al.* present Pharos, a distributed architecture for locating diverse sources of information on the Internet. Such architectures must scale well in terms of information gathering with the increasing diversity of data, the dispersal of information among a growing data volume. Pharos is designed to scale well w.r.t. all these aspects, to beyond 10^5 sources. The use of a hierarchical metadata structure greatly enhances scalability because it features a hierarchical network organization.

In [44] collection selection strategies using cluster-based language models have been investigated. Xu *et al.* proposed three new methods of organizing a distributed retrieval system based on the basic ideas presented before. This three methods are *global clustering*, *local clustering*, and *multiple-topic representation*. In the first method, assuming that all documents are made available in one central repository, a clustering of the collection is created; each cluster is a separate collection that contains only one topic. Selecting the right collections for a query is the same as selecting the right topics for the query. This method is appropriate for searching very large corpora, where the collection size can be in the order of terabytes. The next method is *local clustering* and it is very close to the previous one except the assumption of a central repository of documents. This method can provide competitive distributed retrieval without assuming full cooperation among the subsystems. The disadvantage is that its performance is slightly worse than that of global clustering. The last method is *multiple-topic representation*. In addition to the constraints in local clustering, the authors assume that subsystems do not want to physically partition their documents into several collections. A possible reason is that a subsystem has already created a single index and wants to avoid the cost of re-indexing. However, each subsystem is willing to cluster its documents and summarize its collection as a number of topic models for effective collection selection. With this method a collection corresponds to several topics. Collection selection is based on how well the best topic in a collection matches a query. The advantage of this approach is that it assumes minimum cooperation from the subsystem. The disadvantage is that it is less effective than both global and local clustering.

In this paper we present a novel techniques that is used to simultaneously derive a partitioning strategy and an effective selection function. The novelty in our technique is the use of query logs to extract statistical information that are used to drive partitioning. In turn, the algorithm used to *cluster* the documents produces also a very compact representation of the clusters on which we build our collection selection function.

Query logs have been recently discovered as an effective way to enhance the performance of a web search engine. Recently it has been used to develop effective and efficient policies for caching results [13], [28]. In this paper we exploit query logs in order to extract a description of a document based not on the set of terms it contains (i.e. the bag-of-word model) but, instead, on the set of queries it results to be relevant to.

III. Problem Description

Typically, partitioning and selection strategies are based on information gathered from the document collection. Partitioning, in particular, is either random, or based on clustering documents using k-means [44]. In both cases, documents are partitioned without any knowledge of what queries will be like. We believe that information and statistics about queries may help in driving the partitions to an optimal choice.

Our goal is to cluster the most relevant documents for each query in the same partition. The cluster hypothesis states that *closely associated documents tend to be relevant to the same requests* [42]. Clustering algorithms, like the k-means method cited above, exploit this claim by grouping documents on the basis of their content. We instead based our method on co-clustering queries with the documents returned in reply to each one. The algorithm we adopt is described in [10] and is based on a model exploiting the joint probability of picking up a given couple (q, d), where q is a given query and d is a given document. All these probabilities values are collected into a *contingency matrix*.

Several IR problems are modeled with contingency matrices. For instance, the common vector model for describing documents as vector of terms can be described as a contingency matrix, where each entry is the probability of choosing some term and some document. Obviously, common terms and long documents will have higher probability.

Given a contingency matrix, co-clustering is the general problem of performing simultaneously a clustering of columns and rows, in order to maximize some clustering metrics (e.g. inter-cluster distance). In [10], the authors give a very interesting theoretical model: co-clustering is meant to minimize the loss of information between the original contingency matrix and its approximation given by the co-clustered matrix.

They extend the problem by considering the entries as empirical joint probabilities of two random variables, and they present an elegant algorithm that, step by step, minimizes the loss of information. The algorithm is guaranteed to find an optimal co-clustering, meaning that every other co-clustering with the same number of clusters shows a bigger loss of information.

A. Model of the Problem

As said in the introduction, our goal is to partition the documents of our collection into several clusters, using queries as our driver. This way, documents matching a given query are expected to be located in the same cluster. To do this, we perform co-clustering on the queries from our query-log and the documents from our base.

The way we consider documents for co-clustering can be seen as a new way of modeling documents. So far, two popular ways of modeling documents have been proposed: *bag-of-words*, and *vector space*. Since we know which documents are given as answers to each query, we can represent a document as a *query-vector*.

Query-vector model. Let Φ be a query log containing queries $q_1, q_2, \ldots, q_m$. Let $d_{i1}, d_{i2}, \ldots, d_{in_i}$ be the list of documents returned as results to query q_i. Furthermore, let r_{ij} be the rank value associated to the pair (q_i, d_j). A document d_j is represented as an m-dimensional vector $\delta_j = [\chi_{ij}]^T$, where $\chi_{ij} \in [0, 1]$ is the rank of documents d_j returned as an answer to query q_i. The entries of χ_{ij} are then normalized in order to sum to 1.

Documents that are not hit by any query in the query log are represented by *null query-vectors*. This is a very important feature of our model because it allows us to remove more than half of the documents from the collection without losing precision. This will be described in detail below.

The contingency matrix introduced above, can now be formally defined as $\Upsilon = [\delta_i]_{1 \le i \le n}$ where n is the number of distinct documents that are returned as answers to the queries.[1] For each i, j each entry $\Upsilon_{ij} = r_{ij} / \sum\limits_{i \in D} \sum\limits_{j \in \Phi} r_{ij}$ is the rank of the document d_j for the query q_i normalized so that Υ entries sum up to one. The contingency matrix just defined can be used into the co-clustering algorithm to obtain the document clusters identifying the partitions.

Furthermore, co-clustering considers both documents and queries. We thus have two different kind of results: (i) groups made of documents answering to similar queries, and (ii) groups of queries with similar results. The first kind of results is used to build the document partitioning strategy, while the second is the key to our collection selection strategy (see below).

Before going on with our discussion we would like to show that our model is consistent with the theory supporting co-clustering.

Theorem 1: Let Υ be the contingency matrix in the query-vector model. Let $p(q_i, d_j)$ be the probability of picking up document d_j and query q_j. Then, $\Upsilon_{i,j} = p(q_i, d_j)$

Proof: By basic probability theory we have that $p(q_i, d_j) = p(q_i|d_j) \times p(d_j)$.

$p(q_i|d_j)$ is the probability of picking up q_i by choosing at random among the elements of the vector d_j. The query is not chosen uniformly at random but the probability is the ratio among its normalized relevance r_{ij} and the sum of the normalized rank values contained within d_j.

$$p(q_i|d_j) = \frac{r_{ij}}{\sum\limits_{i \in \Phi} r_{ij}}$$

$p(d_j)$ is, instead the probability of picking up a document d_j by choosing at random among all the documents and by weighting each document according to its general ranking.

$$p(d_j) = \frac{\sum\limits_{i \in \Phi} r_{ij}}{\sum\limits_{i \in D} \sum\limits_{j \in \Phi} r_{ij}}$$

where in all the formulas above r_{ij} is equal to 0 if document d_j is not relevant to query q_i. Also, please note that the denominators are never equal to 0 because we removed the *empty* documents (not answering to any query).

Now, by simply multiplying $p(q_i|d_j)$ by $p(d_j)$, we obtain that

$$p(q_i|d_j) \times p(d_j) = \frac{r_{ij}}{\sum\limits_{i \in \Phi} r_{ij}} \times \frac{\sum\limits_{i \in \Phi} r_{ij}}{\sum\limits_{i \in D} \sum\limits_{j \in \Phi} r_{ij}} = \frac{r_{ij}}{\sum\limits_{i \in D} \sum\limits_{j \in \Phi} r_{ij}}$$

[1] *Empty* documents, i.e. documents never recalled by any query, are removed from the matrix, to speed up the convergence of the algorithm. Only recalled documents are considered.

which is equal to Υ_{ij} by definition. ■

The theorem above is important justifies the use of Υ in the co-clustering algorithm: the working hypothesis for the convergence is thus satisfied.

The result of co-clustering is a matrix $\widehat{P}$ defined as:

$$\widehat{P}(qc_a, dc_b) = \sum_{i \in qc_b} \sum_{j \in dc_a} r_{ij}$$

In other words, each entry $\widehat{P}(qc_a, dc_b)$ sums the contributions of r_{ij} for the queries in the query cluster a and the documents in document cluster b. We call this matrix simply PCAP. The values of PCAP are important because they measure the relevance of a document cluster to a given query cluster. This induces naturally a simple but effective collection selection algorithm.

IV. Our Method

We used the ideas presented in the previous section to design a distributed IR system for Web pages. Our strategy is as follows.

First, we train the system with the query log of the *training period*, by using a reference centralized index to answer all queries submitted to the system. We record the top-ranking results for each query. Then, we perform co-clustering on the query-document matrix. The documents are then partitioned onto several IR cores according to the results of clustering.

We partition the documents into 17 clusters: the first 16 clusters are the clusters returned by co-clustering, and the last one holds the *empty* query-vectors, i.e. the documents that are not returned by any query, represented by null query-vectors.

After the training, we perform collection selection as shown in the next section. The cores holding the selected collections are queried, and results are merged. In order to have comparable document ranking within each core, we distribute the global collection statistics to each IR server. So, the ranking functions are consistent, and results can be very easily merged, simply by sorting documents by their rank.

We keep logging the results of each query, also after the end of the training period, in order to further train the system and also to accommodate any *topic shift*. Topic shift refers to the fact that, over time, the interests of the users of a search engine can change. For instance, in the case of an unexpected calamity, there can be a sudden increase of queries about the issue.

While CORI, among others, perform selection based on term and document statistics, our model is trained to follow the taste of the users: it learns out of the query log. If the range and the topic of queries change substantially over time, it might happen that documents relevant to the new queries are not clustered together, with the net result of a loss of precision if only the first few IR cores are queried.

To adjust to this possibility, the co-clustering algorithm can be periodically performed by an off-line server and documents can be moved from one cluster to another in order to improve precision and reduce server load. One interesting thing is that there is no need for a central server running a centralized version of the search engine because the rank returned by the individual IR cores is consistent with the one a central server would return.

A. Collection Selection

Our selection strategy is based on the PCAP matrix returned by the co-clustering algorithm. The queries belonging to each query cluster are joined together into *query dictionary* files. Each dictionary files stores the text of each query belonging to a cluster, as a single text file. When a new query q is submitted to our IR system, we use the TF.IDF metric to find which clusters are the best matches: each dictionary file is considered as a document, which is indexed with the usual TF.IDF technique. This way, each query cluster qc_i receives a score relative to the query q ($r_q(qc_i)$).

This is used to weight the contribution of PCAP $\widehat{P}(i,j)$ for the document cluster (or IR core) dc_j, as follows:

$$r_q(dc_j) = \sum_i r_q(qc_i) \times \widehat{P}(i,j)$$

The last IR core (#17) is always queried as the last one, because the PCAP matrix chooses only among the 16 clusters returned by co-clustering.

V. Experimental Results

We performed our test using the WBR99 collection. WBR99 consists of 5,939,061 documents documents, about 22 GB uncompressed, representing a snapshot of the Brazilian Web (domains .br) as spidered by www.todobr.com.br. It comprises about 2,700,000 different terms. We could use also the query log of www.todobr.com.br for the period January through October 2003.

Due to the nature of data, we do not have a list of human-chosen relevant documents for each query. WBR99 includes only 50 evaluated queries. Thus, following the example of previous works [43], we consider the top-ranking pages returned by a central index to be relevant. In particular, when measuring precision at 5, we consider only the top five documents to be relevant. Similarly, for precision at 10 and so on.

For our experiment, we used Zettair[2], a compact and fast text search engine designed and written by the Search Engine Group at RMIT University. We modified it so to implement our collection selection strategies (CORI and PCAP).

A. Query-Driven Allocation vs. Random Allocation

Our first experiments were aimed at showing that our document partitioning strategy boosted the results of standard collection selection algorithms. In order to do this, we compared the performance of CORI on our query-driven partition with that on a random document allocation.

In the test, our system was trained with the first three weeks of queries. In other words, we recorded the results of the queries submitted to the system for this period, and we used

[2]Available under a BSD-style license at http://www.seg.rmit.edu.au/zettair/.

Precision at	1	2	4	8	16	17
5	0.3	0.57	1.27	2.62	4.6	5
10	0.59	1.16	2.55	5.0	9.3	10
20	1.20	2.49	5.04	9.77	18.71	20

TABLE I

PRECISION OF CORI ON A RANDOM ALLOCATION, WHEN USING THE FIRST 1, 2, 4, 8, 16 OR 17 CLUSTERS. QUERIES FROM THE FOURTH WEEK.

this to build the query-vector representation of documents and to perform the co-clustering algorithms.

The first three weeks of the log comprise about 190,000 unique queries. We chose to use only unique queries, because the caching system of modern search engines is able, to a certain degree, to avoid issuing the same query multiple times to the system.

The other choice was to consider also repeated queries. This choice would have boosted the relative weights of popular queries in the co-clustering algorithm, with the results of creating a document clustering that favors popular queries. This can be a detrimental choice if caching is actually present. We plan to test this in our future work.

The record of the training queries takes about 130 MB in a binary representation. The co-clustering algorithm took about 10 minutes to run on a single machine.

To speed up our experimental phase, we simulated the document allocation by running Zettair on the full index of documents, and by filtering the results according to our allocation. This let us quickly change the allocation and estimate the results without actually moving the data around. As said before, this strategy does not change the ranking of documents on the IR cores because it is possible to send the collection statistics to each core so to normalize the ranking.

Along with this allocation, we considered a random allocation, where documents are assigned randomly to the servers, an equal share to each one.

We used CORI as our collection selection strategy for the two allocations. We tested it with the queries for the fourth week, i.e. we used the queries for one week following the training period. The test comprise about 194,200 queries. In this case, we considered the statistics of repeated queries, because they represent the effective performance as perceived by user.

On random partition, CORI performs rather poorly (see Figure I). The results are actually very close to what we would get with a random collection selection (0.3 out of 5 is very close to 1 out of 17). This means that, with a poor document partition strategy, CORI does not perform an effective collection selection. It performs dramatically better with a carefully prepared partition (see below).

B. CORI vs. PCAP

In this subsection, we measure the performance of our collection selection strategy w.r.t. CORI. In this case, we test two different allocation strategies on the same document allocation, as generated by the co-clustering algorithm.

CORI Precision at	1	2	4	8	16	17
5	1.57	2.27	2.99	3.82	4.89	5.00
10	3.06	4.46	5.89	7.56	9.77	10.00
20	6.01	8.78	11.64	15.00	19.52	20.00

PCAP Precision at	1	2	4	8	16	17
5	**1.74**	2.30	2.95	3.83	4.85	5.00
10	**3.45**	4.57	5.84	7.60	9.67	10.00
20	**6.93**	9.17	11.68	15.15	19.31	20.00

TABLE II

PRECISION OF THE CORI AND PCAP STRATEGY, WHEN USING THE FIRST 1, 2, 4, 8, 16 OR 17 CLUSTERS. QUERIES FROM THE FOURTH WEEK.

CORI Precision at	1	2	4	8	16	17
5	1.55	2.29	3.01	3.83	4.89	5.00
10	3.05	4.48	5.92	7.62	9.77	10.00
20	5.97	8.77	11.61	15.10	19.54	20.00

PCAP Precision at	1	2	4	8	16	17
5	**1.73**	2.26	2.89	3.76	4.84	5.00
10	**3.47**	4.51	5.75	7.50	9.66	10.00
20	**6.92**	9.02	11.47	14.98	19.29	20.00

TABLE III

PRECISION OF THE CORI AND PCAP STRATEGIES, WHEN USING THE FIRST 1, 2, 4, 8, 16 OR 17 CLUSTERS. QUERIES FROM THE FIFTH WEEK.

In this case, we tested the two selection strategies for the first and the second week after the training. We did this because we wanted to measure the effect of *topic shift* over time. Any difference in precision between the first and the second week gives us a flavor of what the effect of topic shift is. Clearly, we can keep the training going over time: this experiment can suggest us the frequency with which we should refresh our document partitions.

Again, for precision at 5, we consider only the five top-ranking documents (on the full index) to be relevant. Similarly, for precision at 10 we observe the top 10, and so on.

The first experimental results (Tables II and III) show that, in the fourth week (the first after training), PCAP is performing better than CORI: the precision reached with the first cluster is improved of a factor between 11% and 15% (highlighted entries). This is confirmed by the fifth week (the second after training). If present, the topic shift did not affect our results in a significant way.

C. Precision With Top-100 Documents

We wanted to estimate what the net results of our strategy is to users. We measured the precision again, but this time we considered *the 100 top-ranking documents to be relevant*. Table IV describes the results: the results of the first cluster selected by PCAP are more relevant, by a factor of about 10%, than those retrieved by CORI. For the first 20 results are retrieved from the first cluster, we get 13.50 relevant documents with PCAP, while only 12.08 with CORI (11.7%

CORI Precision at	1	2	4	8	16	17
5	4.15	7.57	13.05	21.94	35.60	36.98
10	7.31	12.74	20.54	32.16	48.97	50.73
20	12.08	19.86	29.65	43.35	62.35	64.33

PCAP Precision at	1	2	4	8	16	17
5	4.37	7.40	12.44	21.61	35.75	36.98
10	7.82	12.47	19.53	31.71	48.82	50.73
20	13.50	19.94	28.63	42.96	61.68	64.25

TABLE IV

PRECISION OF THE CORI AND PCAP STRATEGIES. THE 100 TOP-RANKING DOCUMENTS ARE CONSIDERED TO BE RELEVANT.

dc:	number of document clusters	17
qc:	number of query clusters	128
d:	number of documents	5,939,061
t:	number of distinct terms	2,700,000
t':	number of distinct terms in the query dictionary	74,767

TABLE V

STATISTICS ABOUT COLLECTION REPRESENTATION.

better). This means that, when we limit ourselves to only one cluster, the list of results is 10% more precise to the user.

D. Footprint of the Representation

Every collection selection strategy needs a representation of the collections, which is used to perform the selection. We call dc the number of different collections (document clusters), qc the number of query clusters, t the number of terms, t' the number of distinct terms in the query log, d the number of documents, q the number of queries.

CORI representation includes:

- $df_{i,k}$, the number of documents in collection i containing term k, which is $O(dc \times t)$ before compression,
- cw_i, the number of different terms in collection i, $O(dc)$,
- cf_k, the number of resources containing the term k, $O(t)$.

This is in the order of $O(dc \times t) + O(dc) + O(t)$, before compression.

On the other side, the PCAP representation is composed of:

- the PCAP matrix, with the computed $\widehat{p}$, which is $O(dc \times qc)$,
- the index for the query clusters, which can be seen as $n_{i,k}$, the number of occurences of term k in the query cluster i, for each term occurring in the queries — $O(qc \times t')$.

This is in the order of $O(dc \times qc) + O(qc \times t')$, before compression. This is significantly smaller than the CORI representation as $dc << d$, $t' << t$, and $qc << t$ (see Table V).

E. Empty Documents

At the end of the training period, we observed that a large number of documents, around 52% of them (3,128,366), are not returned among the first 100 top-ranked results of any query. This means that, during the training period, no user was invited to open these documents as a result of his/her query (unless s/he browser beyond the 100-th result).

Separating these documents from the rest of the collection allows the indexer to produce a more compact index, containing only relevant documents that are likely to be requested in the future. This pruning process can speed up the overall performance significantly.

The pruned documents can be stored in another server, used only when the user browse for low-relevance documents. In our test, we showed that the contribution to precision of PCAP is very low for the last cluster. In Tables II and III, the change in precision when adding also the last cluster (#17) is in the range of 2%–3%. For instance, the precision at 5 for the fourth week passes from 4.85 to 5.

More dramatically said, less than 50% of the documents contribute more than 97% of the relevant documents.

VI. CONCLUSION AND FUTURE WORK

In this work, we presented a novel approach to document partitioning and collection selection, based on co-clustering queries and documents. Our new representation of documents as query-vectors allows us to perform, very efficiently, an effective partitioning. It also induces a very compact representation of the resulting collections. We showed that our selection strategy out-performed CORI by a factor of 10%, with a smaller representation of the available collections.

Also, we showed a way to select rarely asked-for documents. The process of pruning these documents could improve the performance of the system, because less than 50% contributes more than 97% of the relevant documents: around 52% of the documents (3,128,366) are not returned among the first 100 top-ranked results of any query.

We would like to compare our approach with other partitioning strategies. While the goals are not common (load balancing, collection selection...), we believe our work adds to the existing results on document and term partitioning.

We would like to explore, with more experiment, the impact of our design choice on the final results.

ACKNOWLEDGMENT

The authors would like to thank Raffaele Perego, Claudio Lucchese and Salvatore Orlando for the fruitful discussions around the coffee machine. Raffaele is also the author of Figure 1.

This work was partially supported by the Italian MIUR FIRB Grid.it project (RBNE01KNFP) on High-Performance Grid Platforms and Tools, and by the MIUR CNR strategic Project L499/1997-2000 on High-Performance Platforms.

REFERENCES

[1] Claudine Santos Badue, Ramurti Barbosa, Paulo Golgher, Berthier Ribeiro-Neto, and Nivio Ziviani. Distributed processing of conjunctive queries. In *HDIR '05: Proceedings of the First International Workshop on Heterogeneous and Distributed Information Retrieval (HDIR'05)*, SIGIR 2005, Salvador, Bahia, Brazil, 2005.

[2] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] S. Brin and L. Page. The Anatomy of a Large–Scale Hypertextual Web Search Engine. In *Proceedings of the WWW7 conference / Computer Networks*, volume 1–7, pages 107–117, April 1998.

[4] Fidel Cacheda and Vassilis Plachouras. Performance analysis of distributed architectures to index one terabyte of text. volume 2997, pages 394–408, 2004/01//.

[5] Brendon Cahoon, Kathryn S. McKinley, and Zhihong Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM Trans. Inf. Syst.*, 18(1):1–43, 2000.

[6] Jamie Callan and Margaret Connell. Query–Based Sampling of Text Databases.

[7] J.P. Callan, Z. Lu, and W.B. Croft. Searching Distributed Collections with Inference Networks. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 21–28, Seattle, WA, July 1995. ACM Press.

[8] J. Callan, A. Powell, J French, and M. Connell The Effects of Query–Based Sampling on Automatic Database Selection Algorithms. In *Technical Report CMU-LTI-00-162*, Language Technologies Institute, School of Computer Science, Carnegie Mellon University. 2000.

[9] N. Craswell, P. Bailey, and D. Hawking. Server Selection on the World Wide Web. In *Proceedings of the Fifth ACM Conference on Digital Libraries*, pages 37–46, 2000.

[10] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2003)*, pages 89–98, 2003.

[11] R. Dolin, D. Agrawal, L. Dillon, and A. El Abbadi. Pharos: A Scalable Distributed Architecture for Locating Heterogeneous Information Sources. Technical Report TRCS96–05, 5 1996.

[12] Christoph Baumgarten Email. A Probabilistic Solution to the Selection and Fusion Problem in Distributed Information Retrieval.

[13] T. Fagni, F. Silvestri, S. Orlando, and R. Perego. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM TOIS (Transactions on Information Systems)*, 2006. To appear.

[14] James C. French, Allison L. Powell, James P. Callan, Charles L. Viles, Travis Emmitt, Kevin J. Prey, and Yun Mou. Comparing the Performance of Database Selection Algorithms. In *Research and Development in Information Retrieval*, pages 238–245, 1999.

[15] James C. French, Allison L. Powell, and Jamie Callan. Effective and Efficient Automatic Database Selection. Technical Report CS-99-08, 2 1999.

[16] James C. French, Allison L. Powell, Charles L. Viles, Travis Emmitt, and Kevin J. Prey. Evaluating Database Selection Techniques: A Testbed and Experiment. In *Research and Development in Information Retrieval*, pages 121–129, 1998.

[17] N. Fuhr. A Decision–Theoretic Approach to Database Selection in Networked IR. In Workshop on Distributed IR, 1996.

[18] N. Fuhr. Optimum Database Selection in Networked IR. In *Proceedings of the SIGIR'96 Workshop Networked Information Retrieval*, Zurich, Switzerland, 1996.

[19] L. Gravano and H. Garcia–Molina. Generalizing GlOSS to Vector–Space Databases and Broker Hierarchies. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, 1995.

[20] L. Gravano, H. Garcia–Molina, and A. Tomasic. Precision and Recall of GlOSS Estimators for Database Discovery. Stanford University Technical Note Number STAN-CS-TN-94-10, 1994.

[21] L. Gravano, H. Garcia–Molina, and A. Tomasic. The Effectiveness of GlOSS for the Text Database Discovery Problem. In *Proceedings of the SIGMOD 94*. ACM, September 1994.

[22] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. The Efficacy of GlOSS for the Text Database Discovery Problem. Technical Report CS-TN-93-2.

[23] David Hawking and Paul Thistlewaite. Methods for Information Server Selection. *ACM Transactions on Information Systems*, 17(1):40–76, 1999.

[24] Paul Ogilvie Jamie. The Effectiveness of Query Expansion for Distributed Information Retrieval.

[25] Trevor Jim and Dan Suciu. Dynamically Distributed Query Evaluation. In *Proceedings of the PODS 2001 conference*, Santa Barbara, CA, May 2001. ACM.

[26] Donald Kossmann. The State of the Art in Distributed Query Processing. *ACM Computer Survey*, 1999.

[27] Leah S. Larkey, Connel, Margaret, and Jamie Callan. Collection selection and results merging with topically organized u.s. patents and trec data. To appear in Proceedings of Ninth International Conference on Information Knowledge and Management, November 6–10 2000.

[28] Ronny Lempel and Shlomo Moran. Predictive caching and prefetching of query results in search engines. In *Proc. of the twelfth international conference on World Wide Web*, pages 19–28. ACM Press, 2003.

[29] Mei Li, Wang-Chien Lee, and Anand Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. *12th IEEE International Conference on Network Protocols (ICNP'04)*, 2004.

[30] Z. Lu, J. Callan, and W. Croft. Measures in Collection Ranking Evaluation, 1996.

[31] Zhihong Lu and K.S. McKinley. *Advances in Information Retrieval*, chapter 7. The Effect of Collection Organization and Query Locality on Information Retrieval System Performance and Design. Kluwer, New York. Bruce Croft Editor, 2000.

[32] A. MacFarlane, J. A. McCann, and S. E. Robertson. Parallel search using partitioned inverted files. In *SPIRE '00: Proceedings of the Seventh International Symposium on String Processing Information Retrieval (SPIRE'00)*, page 209, Washington, DC, USA, 2000. IEEE Computer Society.

[33] Alistair Moffat and Justin Zobel. Information Retrieval Systems for Large Document Collections. In *Proceddings of the Text REtrieval Conference*, pages 85–94, 1994.

[34] Salvatore Orlando, Raffaele Perego, and Fabrizio Silvestri. Design of a Parallel and Distributed WEB Search Engine. In *Proceedings of Parallel Computing (ParCo) 2001 conference*. Imperial College Press, September 2001.

[35] A. Powell, J. French, J. Callan, Connell, and C. M. Viles. The Impact of Database Selection on Distributed Searching. In *Proceedings of the SIGIR 2000 conference*, pages 232–239. ACM, 2000.

[36] Anand Rajamaran and Jeffrey D. Ullman. Querying Websites Using Compact Skeletons. In *Proceedings of the PODS 2001 conference*, Santa Barbara, CA, May 2001. ACM.

[37] Yves Rasolofo. Approaches to Collection Selection and Results Merging for Distributed Information Retrieval.

[38] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[39] Yipeng Shen, Dik Lun Lee, and Lian Wen Zhang. A Distributed Search System Based on Markov Decision Processes. In *Proceedings of the ICSC 99 conference*, Honk Kong, December 1999.

[40] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks, 2002.

[41] Anthony Tomasic, Luis Gravano, Calvin Lue, Peter Schwarz, and Laura Haas. Data Structures for Efficient Broker Implementation. *ACM Transactions on Information Systems*, 15(3):223–253, 1997.

[42] C.J. Van Rijsbergen. *Information Retrieval*. Butterworths, 1979.

[43] Xu, Jinxi, and W.B. Croft. Effective Retrieval with Disributed Collections. In *Proceedings of SIGIR98 conference*, Melbourne, Australia, August 1998.

[44] Jinxi Xu and W. Bruce Croft. Cluster-Based Language Models for Distributed Retrieval. In *Research and Development in Information Retrieval*, pages 254–261, 1999.

[45] Budi Yuwono and Dik Lun Lee. Server Ranking for Distributed Text Retrieval Systems on the Internet. In *Database Systems for Advanced Applications*, pages 41–50, 1997.