Collection Selection
...now, with more documents!

Diego Puppin
diego.puppin@alum.mit.edu[*]

ABSTRACT
A way to reduce the computing pressure in a distributed IR system is to use document partitioning and to perform collection selection. With suitable training and/or modeling, the collection selection function can choose the most promising collections for each query, with high confidence. Unfortunately, if the collections need to be updated, we need to retrain the selection function, update its statistics or face the loss of some result quality. This paper introduces a simple, but very effective, technique to add new documents to collections in a system that uses collection selection. We show that we can update the individual collections, while guaranteeing the same selection performance, with no need to update or retrain the selection function.

Keywords
Distributed IR, Collection Selection, Index Update, Web Search Engines

1. CAPTATIO BENEVOLENTIÆ
The Web has now become the place where million people look, daily, for entertainment, information, shopping opportunities, social interaction. About 1.1 billion people have access to the Internet, and Web browsing is, by far, the most popular activity. 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.

Moreover, the Web is changing faster and faster. Experienced users have come to expect very fresh content on their preferred blogs, and to expect search engines to react fast to changes in their most loved Web sites. Successful search engines have to crawl, index and search quickly billions of pages, for millions of users every day. They also need to update their document base with rapidity, so that the latest content can be easily found by queries.

A modern Web search engine is implemented using a large pool of computing servers, which share the engine's large index. When each server holds the part of the index 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.

[*]This study was completed while the author was working as a researcher at ISTI-CNR (Pisa, Italy). The author currently works as a software engineer at Google (Boston, USA), which supported his participation to the conference.

Document partitioning is the strategy chosen by the most popular Web search engines. In the document-partitioned organization, the broker may choose among two possible strategies for scheduling a query. A simple, 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 good, yet not perfect, load balancing among the servers [Badue et al. 2007]. On the other hand, it has the major drawback of utilizing every server for each query submitted, this way causing a higher overall computing load.

The other possible way of scheduling is to choose, for each query, the most authoritative server(s). By doing so, we reduce the number of queried collections. The relevance of each server to a given query is computed by means of a selection function that can be built upon statistics computed over each sub-collection. This process, called *collection selection*, is considered to be an effective technique to enhance the capacity of distributed IR systems [Baeza-Yates et al. 2007].

In [Puppin et al. 2006, Puppin and Silvestri 2006], we presented a novel architecture for a distributed search engine, based on collection selection. We exploit the knowledge about the past use of the search engine to drive the assignment of documents to sub-collections, and to choose the most promising sub-collections for each query. Our architecture relies on effective strategies for load balancing and result caching, and aims at returning very high-quality results by using only a small and tunable fraction of the computing load of a traditional document-partitioned search engine.

In this paper, we show how to update the document base without affecting the performance of our collection selection algorithm. A feature of our PCAP architecture is that it is trained with data from the query log, and new documents need to be carefully placed among collection so that they can be easily found by our selection algorithm. Our simple algorithm allows us to select where to position a new document so that sub-collections are still consistent and of high quality. Its modest computing cost makes it suitable for a fast and continuous utilization, while offering a performance similar to that reached by reassigning all documents from start.

The paper is organized as follows. In the next section, we summarize the main features of our PCAP architecture. Then, we discuss previous results on index updating. We introduce our technique in Section 4, which is followed by a summary of our experimental results. Finally, we part from our reader with some final comments and plans for future work.

2. THE PCAP ARCHITECTURE

2.1 The QV Document Model

In the QV (*query-vector*) model, documents are represented by the weighted list of queries (out of a training set) that recall them: the QV representation of document d is a vector where each dimension is the score that d gets for each query in the query set. The set of the QVs of all the documents in a collection can be used to build a query-document matrix, which can be normalized and considered as an empirical joint distribution of queries and documents in the collection. This allows us to co-cluster queries and documents, so to identify queries recalling similar documents, and groups of documents related to similar queries. The algorithm we adopt is described in [Dhillon et al. 2003], and is based on a model exploiting the empirical joint probability of picking up a given couple (q, d), where q is a given query and d is a given document. The results of the co-clustering algorithm are then used to perform collection selection.

A reference search engine is used in the building phase: for every query in the training set, the system stores the first N results along with their rank. Figure 1 gives an example. The first query $q1$ recalls, in order, $d3$ with rank 0.8, $d2$ with rank 0.5 and so on. Query $q2$ recalls $d1$ with rank 0.3, $d3$ with rank 0.2 and so on. We may have empty columns, when a document is never recalled by any query (in this example $d5$). Also, we can have empty rows when a query returns no results ($q3$).

Formally, let Q 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, by a reference search engine, as results to query q_i. Furthermore, let r_{ij} be the score that document d_j gets as result of query q_i (0 if the document is not a match).

A document d_j is represented as an m-dimensional *query-vector* $\overline{d}_j = [\overline{r_{ij}}]^T$, where $\overline{r_{ij}} \in [0, 1]$ is the normalized value of r_{ij}:

$$\overline{r_{ij}} = \frac{r_{ij}}{\sum\limits_{i \in Q} \sum\limits_{j \in D} r_{ij}}$$

The QV model will be simply built out of the results recalled by the engine using a given query log. Our system can also identify *silent documents*, *i.e.* documents never recalled by any query from the query log Q. Silent documents are represented by *null query-vectors*. The ability of identifying silent documents is a very important feature of our model because it allows us to determine a set of documents that can safely be moved to a supplemental index.

Query/Doc	d1	d2	d3	d4	d5	d6	...	dn
q1	-	0.5	0.8	0.4	-	0.1	...	-
q2	0.3	-	0.2	-	-	-	...	0.1
q3	-	-	-	-	-	-	...	-
q4	-	0.4	-	0.2	-	0.5	...	0.3
...	...	...	...	...	...	...	...	...
qm	0.1	0.5	0.8	-	-	-	...	-

Figure 1: In the query-vector model, every document is represented by the query it matches (weighted with the rank).

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

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

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

In other words, each entry $\widehat{P}(qc_a, dc_b)$ sums the contributions of $\overline{r_{ij}}$ for the queries in the query cluster a and the documents in document cluster b. We call this matrix simply PCAP, from the LaTeX command `\cap{P}` needed to typeset it.[1] 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.

2.2 PCAP Selection Algorithm

The queries belonging to each query cluster are chained together into *query dictionary* files. Each dictionary file stores the text of each query belonging to a cluster, as a single text file. For instance, if the four queries *hotel in Texas*, *resort*, *accommodation in Dallas* and *hotel downtown Dallas Texas* are clustered together as the first query cluster, the first query dictionary will simply be qc_1=*"hotel in Texas resort accommodation in Dallas hotel downtown Dallas Texas"*. The second query dictionary file could be, for instance qc_2 =*"car dealer Texas buy used cars in Dallas automobile retailer Dallas TX"*. A third query cluster could be qc_3 =*"restaurant chinese restaurant eating chinese Cambridge"*.

When a new query q is submitted to the IR system, the BM25 metric [Robertson and Walker 1994] is used to find which query clusters are the best matches: each dictionary file is considered as a document, which is indexed using the vector-space model, and then queried with the usual BM25 technique. This way, each query cluster qc_i receives a score relative to the query q, say $r_q(qc_i)$. In our example, if a user asks the query *"used Ford retailers in Dallas"*, $r_q(qc_2)$ will be higher than $r_q(qc_1)$ and $r_q(qc_3)$.

[1]The correct LaTeX command for $\widehat{P}$ is actually `\widehat{P}`, but we initially thought it was `\cap{P}`, which incorrectly gives $\cap P$. Hats, caps... whatever.

PCAP	dc1	dc2	dc3	dc4	dc5	$r_q(qc_i)$
qc1		0.5	0.8	0.1		0.2
qc2	0.3		0.2		0.1	0.8
qc3	0.1	0.5	0.8			0

$$\begin{array}{lcrcrcrcl}
r_q(dc_1) & = & 0 & + & 0.3 \times 0.8 & + & 0 & = & 0.24 \\
r_q(dc_2) & = & 0.5 \times 0.2 & + & 0 & + & 0 & = & 0.10 \\
r_q(dc_3) & = & 0.8 \times 0.2 & + & 0.2 \times 0.8 & + & 0 & = & 0.32 \\
r_q(dc_4) & = & 0.1 \times 0.2 & + & 0 & + & 0 & = & 0.02 \\
r_q(dc_5) & = & 0 & + & 0.1 \times 0.8 & + & 0 & = & 0.08
\end{array}$$

Figure 2: Example of PCAP to perform collection selection. We have three query clusters: qc_1=*"hotel in Texas resort accommodation in Dallas hotel downtown Dallas Texas"*, qc_2 =*"car dealer Texas buy used cars in Dallas automobile retailer Dallas TX"* and qc_3 =*"restaurant chinese restaurant eating chinese Cambridge"*. The second cluster is the best match for the query *"used Ford retailers in Dallas"*. The third document cluster is expected to have the best answers.

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

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

Figure 2 gives an example. The top table shows the PCAP matrix for three query clusters and five document clusters. Suppose BM25 ranks the query-clusters respectively 0.2, 0.8 and 0, for a given query q. We compute the vector $r_q(dc_i)$ by multiplying the matrix PCAP by $r_q(qc_i)$, and we will choose the collection dc_3, dc_1, dc_2, dc_5, dc_4 in this order.

The QV model and the PCAP selection function together are able to create very robust document partitions. In addition, they allow the search engine to identify, with great confidence, what the most authoritative servers are for any query. Experimental results are available in [Puppin et al. 2006, Puppin and Silvestri 2006].

In [Puppin et al. 2007] we introduce load-driven routing and incremental caching, two techniques able to address differences in query pressure on individual servers. While these two algorithms are not central to the topic of this paper, they are an important feature of the reference architecture, fundamental to reach a good load balance across servers.

3. STANDING ON THE SHOULDERS OF GIANTS

The algorithms to keep the index of an IR system up to date are very complex, and have been the interest of several works in the scientific literature.

In [Arasu et al. 2001], the opening article on the first issue of ACM TOIT (2001), the authors comment on the open problems of Web search engines, and clearly identify the update of the index as a critical point. They suggest to rebuild the index after every crawl, because most techniques for incremental index update do not perform very well when they have to deal with the huge changes (in term of the number of documents added, removed and changed) that are commonly observed between successive crawls of the Web. The paper [Melink et al. 2001] (cited in [Arasu et al. 2001]) gives a measure of the phenomenon, and emphasizes the need to perform research in this area. The opportunity of a batch update of the index (rather than incremental) is also supported by a previous work [Frieder et al. 2000], which show that the result precision is not affected by a delayed update of the index.

The problem is made more complex in distributed system. In a term-partitioned system, when a document is added, changed or removed, we need to update the index in all servers holding at least one term from the document. While this update can be fast because the update is executed in parallel in several locations of the index (several different posting lists), the communication costs are very high, and they make this approach unfeasible. On a document-partitioned system, instead, all the changes are local to the IR server to which the document is assigned: when a document is assigned to one server, only the local index of that server needs to be changed. This is another reason why document-partitioned systems are preferred to term-partitioned systems in commercial installations [Brin and Page 1998, Baeza-Yates et al. 2007].

One of the first papers to address the incremental update of an index is [Tomasic and Garcia-Molina 1993], which, in 1993, presented a a dual-structure index to keep the inverted lists up to date. The proposed structure dynamically separated long and short inverted lists and optimized retrieval, update, and storage of each type of list. The authors show an interesting trade-off between optimizing update time and optimizing query performance. The result is a starting point for later papers, which explicitly address a very high rate of change.

In [Lim et al. 2003], the authors acknowledge that, while results in incremental crawling have enabled the indexed document collection of a search engine to be more synchronized with the changing World Wide Web, the new data are not immediately searchable, because rebuilding the index is very time-consuming. The authors explicitly study how to update the index for changed documents. Their method uses the idea of *landmarks*, *i.e.* marking some important spots in the inverted lists, together with the *diff* algorithm, to significantly reduce the number of postings in the inverted index that needs to be updated.

Another important result was presented in [Lester et al. 2004]. The authors dicuss the three main alternative strategies for index update: in-place update, index merging, and complete re-build. The experiments showed that re-merge is, for large numbers of updates, the fastest approach, but in-place update is suitable when the rate of update is low or buffer size is limited. The availability of algorithms targeted to different speed of change is important because, in a document-partitioned system, different collections could change at a different speed.

A very different approach to update is analyzed in [Shieh and Chung 2005]. Following previous literature, the authors suggest to book some sparing free space in an inverted file, so to allow for incremental updates. They propose a run-time statistics-based approach to allocate the spare space. This approach estimates the space requirements in an inverted file using only some recent statistical data on space usage and document update request rate. Clearly, there is a trade-off between the cost of an update and the extra storage needed by the free space. They show that the technique can greatly reduce the update cost, without affecting the access speed to the index.

This paper tackles a complementary problem. In our document-partitioned architecture, when a new document (or a new version of a known document) becomes available, we need to first determine the collection to which it should be assigned. If the assignment is incorrect, the new document will be lost among unrelated information, and the collection selection function will not be able to find it. When the assignment is performed, the system can use all the standard techniques to update the local slice of the index.

The size of the individual collection has a very strong impact on the performance of the system. Several works have proposed different cost model for queries. In his work on large English text corpora, Heaps [Heaps 1978] showed that as more instance text is gathered, there will be diminishing returns in terms of discovery of the full vocabulary from which the distinct terms are drawn. This law was verified also for pages coming from the Web. The law can be restated by saying that the number of distinct terms $T(n)$ from a collection of n documents grows as:

$$T(n) = K \cdot n^{\beta}$$

with K between 10 and 100, and β between 0.4 and 0.6 for collection of English language documents. If documents are of average size S:

$$\text{total number of terms} = n \cdot S$$

$$\text{average size of a posting list} = \frac{n \cdot S}{K \cdot n^{\beta}} = \frac{S}{K} n^{1-\beta}$$

According to the Heaps law, the growth of the average posting list is sub-linear. If the length of the posting lists drives the cost of solving a query, this will grow slower than the growth of the collection.

[Chowdhury and Pass 2003] models the response time as linearly dependent from the collection size. While this is only a first-order modeling of costs, as discussed in [Badue et al. 2007], it is very important to keep collections reasonably balanced, in order to avoid a great disparity in the computing pressure to servers. We will show that our strategy is able to keep the collection size comparable, so that the cost of answering each query is comparable among the servers.

4. UPDATING THE INDEX: A MODEST PROPOSAL

Several researchers have investigated the problem of keeping the document base and the index up to date, in a changing environment like the Web. Two main issues have been investigated. First, the IR system needs to plan the crawling strategy so to find new pages, to identify new versions of pages and to prevent frequently changing pages (*e.g.* news sites, blogs) from becoming stale in the index. Second, the new documents, or their new versions, must be parsed and indexed, and the main index must be updated.

In a distributed, document-partitioned configuration, new documents are added to one of the IR cores. The core must extract the posting lists for the new pages and make them searchable. One simple way to do it is to create a second index with the new postings, and query it in parallel with the main index. The second, smaller index can be updated when new documents are available and, after a chosen time interval, the main and the second index can be merged into one. The server undergoing this merging procedure may become unavailable to answer queries, but this is usually not problematic if several replicas are deployed to provide enough computing bandwidth.

If the query is broadcasted to all cores, as in traditional document-partitioned systems, the choice of the core to which a new document is mapped does not have any effect on the precision of the system. It can have an impact on performance if the partitions are heavily unbalanced, but this can be simply prevented by keeping track of the collection size in each server.

In our architecture, on the other side, the choice is more important: it is crucial to assign new documents to the *correct* collection, *i.e.* to the server where most similar or related documents are already assigned. In other word, we want to store the new document where the collection selection function will find it. This is complex because the document could be relevant for several queries, and the collections will have a different relevance or authoritativeness for each of them.

An ideal approach would be as follows. First, all queries from the training set are performed against the new documents. This way, we can determine which documents are matching which queries, and create the QV representation for the new documents. The vectors are added to the QV vectors of all the other existing documents, and co-clustering is performed on the new matrix (using the previous clustering as a starting point). This is clearly unfeasible, as the training set can be composed of several thousand queries.

A very quick approximation can be reached using the PCAP collection selection function itself. *The body of the document can be used in place of a query: the system will rank the collections according to their relevance.* The rationale is simple: the terms in the document will find the servers holding all the other documents relative to the same broad topics. Our collection selection function is able to rank the document collections according to the relevance to a given query. If the body of the document is used as query, the collection selection function will find the closest collections.

In the first phase, we perform a query, using the document body as a topic, against the query dictionaries, and we determine the matching query clusters, *i.e.* the clusters comprising query with terms appearing also in the document. Then, the PCAP matrix is used to choose the best document clusters, starting from the query cluster ranking, as seen in Section 2.2. If no query dictionary matches the new document, it will be classified as *silent* and stored in the supplemental index.

Note that this process is computing, in an approximate way, the set of queries that could recall a new document. By comparing the document body with the query dictionaries, it tries to understand what query topic could retrieve the given document. This is somewhat similar to what is done by pSearch [Tang et al. 2002] and SSW [Li et al. 2004]. These two systems project the documents onto a lower dimensional space using LSI, and then route them to the correct servers using the same infrastructure used by the queries.

Even if we will prove that this approach is robust, there are several issues that could make it fail. For example, some query cluster could be composed of queries for which the document is actually a good matching, but the terms of which do not appear in the document itself. This can happen for instance in the presence of synonyms, or if the document itself uses graphics to represent words and the query terms are used, instead, in links pointing to the page.

This problem is limited by the redundancy of the QV matrix and the robustness of the algorithm, as we discussed in our previous works [Puppin et al. 2006, Puppin and Silvestri 2006]. Even if one good query, in some query clusters, may use terms not present in the document, the query clusters will be comprised of queries retrieving close documents, and those terms might be present in the document. In any case, after the document is mapped to one cluster, a full search algorithm (using links, PageRank etc.) is used on the local index, and the document will be correctly retrieved and ranked.

From the moment of the assignment on, the new document will appear among the results of the new queries. They will contribute to building a new QV matrix, used when the system chooses to have a major update of the whole index. In fact, every once in a while, the system will be momentarily stopped to update the partitioning and the local indexes. While the latter is needed in every document-partitioned architecture, our collection selection architecture also need to update the assignment of documents to clusters. This means moving a document from its initial position to the position suggested by a complete round of co-clustering. We will verify below that this is not strictly necessary, as the approximate choice is good enough to our purposes.

5. THIS ACTUALLY WORKS

To test our approach, we used the WBR99 Web document collection[2], of 5,939,061 documents, *i.e.* Web pages, representing a snapshot of the Brazilian Web (domains .br) as spidered by the crawler of the TodoBR search engine. The collection consists of about 22 GB of uncompressed data, mostly in Portuguese and English, and comprises about 2.7 million different terms after stemming.

Along with the collection, we used a query log of queries submitted to TodoBR, in the period January through October 2003. We selected the first three weeks of the log as our training set. It is composed of about half a million queries, of which 190,000 are distinct. Then, we used subsequent weeks as test sets. Every week comprises about 200,000 queries.

[2]Thanks to Nivio Ziviani and his group at UFMG, Brazil, who kindly provided the collection, along with logs and evaluated queries.

d	5,939,061 documents taking (uncompressed) 22 GB
t	2,700,000 unique terms
t'	74,767 unique terms in queries
tq	494,113 (190,057 unique) queries in the training set

Table 1: Main feature of our test set.

The main features of our setup are summarized in Table 1. To index and query each sub-collection, we used Zettair[3], a compact and fast text search engine designed and written by the Search Engine Group at RMIT University.

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

We partitioned the first 5,000,000 documents into 17 clusters: the first 16 clusters are the clusters returned by co-clustering; the last one holds the silent documents, *i.e.* the documents that are not returned by any query, represented by null query-vectors (the 17-th cluster is used as a supplemental index). We chose to partition into 16 clusters because a smaller number of document clusters would have brought to a situation with a very simple selection process, while a bigger number would have created artificially small collections.

We assumed that a part of the document collection, about one million documents, is not initially available, because it represents newer or changed pages collected over the two weeks following the end of the initial training. After the training, we use the strategy suggested above to assign the new documents as they arrive. To speed the process up, only the first 1000 bytes from the document body (stripped of HTML tags, header and so on) were used as a query to determine the best collection assignment.

We simulated six different configurations, as shown in Figure 3:

A. This represents the situation right after the training, with 5 million documents available, partitioned using co-clustering. We test against the first set of test queries (queries for the first week after the training).

B. This represents the situation one week later, with 5.5 million documents available. Documents are assigned using PCAP as just described. We test against the second set of test queries (queries for the second week after the training).

C. This is similar to B, but now old and new documents are clustered using a full run of co-clustering. Same test set.

D. This is the situation after two weeks, with all documents available. In this configuration, all new documents are

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

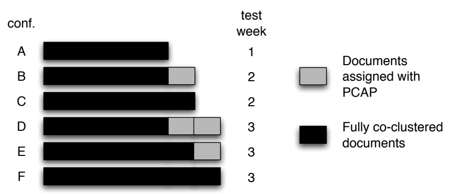

Figure 3: Different configurations for the experiment on index update.

assigned with PCAP. We use the third week of queries.

E. Here we imagine that we were able to run co-clustering for the first half million new documents, but not for the latest ones. Same test set as D.

F. Finally, all 6 million documents are clustered with co-clustering. Same test set as D.

5.1 Partition Sizes

The first important aspect to consider when updating the partitions is their relative sizes: if a partition grows much bigger than another one, the server holding it could become slower than the other ones, contributing to a system-wide degradation of performance. Even if, in our architecture, this problem is limited by the use of a load-driven collection selection strategy (see [Puppin et al. 2007]), still we want to verify if this can be a problem or not.

We measured the size of the different partitions in each configuration. Results are shown in Figure 4. The plots show the size of the collection held by each cluster. The size of the collection held by the overflow cluster is beyond scale and truncated. The horizontal line shows the average size for the clusters, excluding the overflow cluster.

In the worst case, *i.e.* Configuration D, the biggest cluster holds about 2.5 times the documents held in the smallest cluster. In the other cases, this difference is a little bigger than 2.1 times, and the distance from the average is not dramatic. It is not surprising that Configuration D has the worst distribution: it is the configuration with the biggest number of documents not assigned using co-clustering.

According to the Heaps law, the average size of posting lists, in a collection 2.1 times bigger, is about 1.3–1.5 times bigger. We showed that the load-driven collection selection is able to effectively distribute the load across servers, and that a local cache of posting lists is extremely effectively: this disparity is more than tolerable by our system.

5.2 Result Quality

To compare the various strategy, we ask the collection selection function to choose a fixed number of servers, and we verify how many of the relevant documents are present in the chosen servers. In our experiments, lacking human-evaluated reference results, we follow the example of previous works [Xu et al. 1998], and we compare the results coming from collection selection with the results coming from a centralized index, which form the set of reference results. We indexed the full set of documents with Zettair, and we

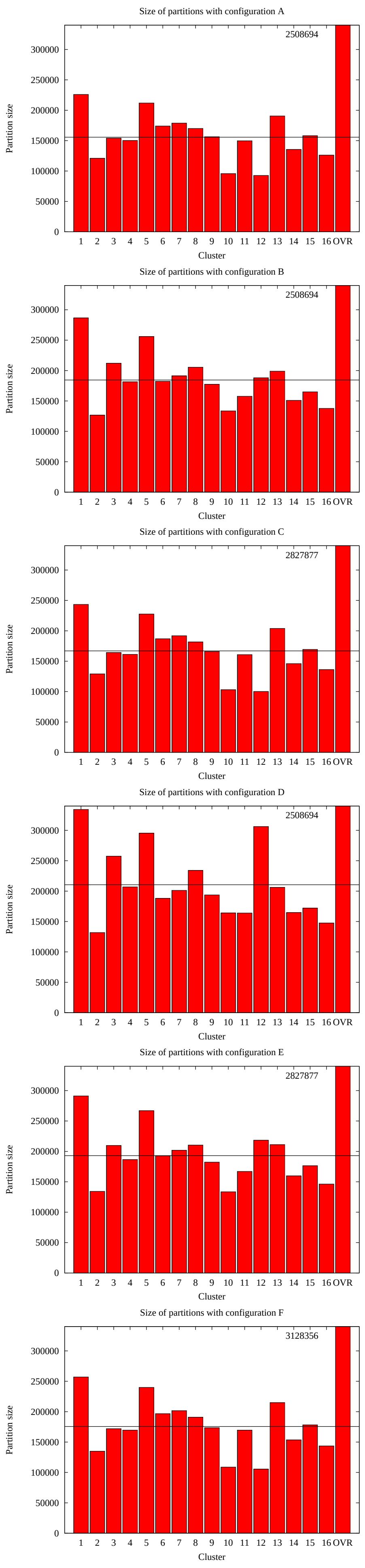

Figure 4: Size of the document collections in each cluster.

used it as our golden standard. This way, we are comparing the same search algorithm (Zettair) on a centralized index w.r.t. an index partitioned across several servers.

We use two metrics: *intersection* and *competitive similarity*. Let's call G_q^N the top N results returned for q by a centralized index (*ground truth*), and H_q^N the top N results returned for q by the set of servers chosen by our collection selection strategy. The *intersection at N*, $INTER_N(q)$, for a query q is the fraction of results retrieved by the collection selection algorithm that appear among the top N documents in the centralized index:

$$INTER_N(q) = \frac{|H_q^N \cap G_q^N|}{|G_q^N|}$$

Given a set D of documents, we call *total score* the value:

$$S_q(D) = \sum_{d \in D} r_q(d)$$

with $r_q(d)$ the score of d for query q. The competitive similarity at N $COMP_N$ is measured as:

$$COMP_N(q) = \frac{S_q(H_q^N)}{S_q(G_q^N)}$$

This value measures the relative quality of results coming from collection selection with respect to the best results from the central index. In both cases, if $|G_q^N| = 0$ or $S_q(G_q^N) = 0$, the query q is not used to compute average quality values.

As the reader can see in Figure 5, the collection selection function performs with the same effectiveness on every configuration: the quality of the results coming from the selected servers is not degraded if documents are added with our strategy.

Interestingly enough, the new process is able to reduce the number of silent documents: in Configuration F (all documents fully co-clustered) about 3,100,000 are classified as silent, and stored in the supplemental index, while in Configuration D (5,000,000 documents co-clustered, 1,000,000 documents assigned with PCAP), only 2,500,000 end up in the supplemental index.

This is due to two main reasons. First, we do not have to face the cut-off we had in training: during co-clustering only the top 100 results are used in training, but now even a weak matching is a good drive. Second, and most important, the query clusters, created by co-clustering are already focused and coherent, and allow us to better discriminate documents.

Note that this does not necessarily contribute to result quality: in [Puppin et al. 2006] we showed that the supplemental index count to only 3% results. Still, it could help to have a better balance among partitions. We plan to verify with more experiments the effect that this has on performance and quality, but this actually seems to suggest a way to assign silent documents after co-clustering.

6. ALMOST OVER...

In this work, we presented a simple strategy to assign new documents to collections in a distributed Web search architecture, based on the concept of collection selection. Our

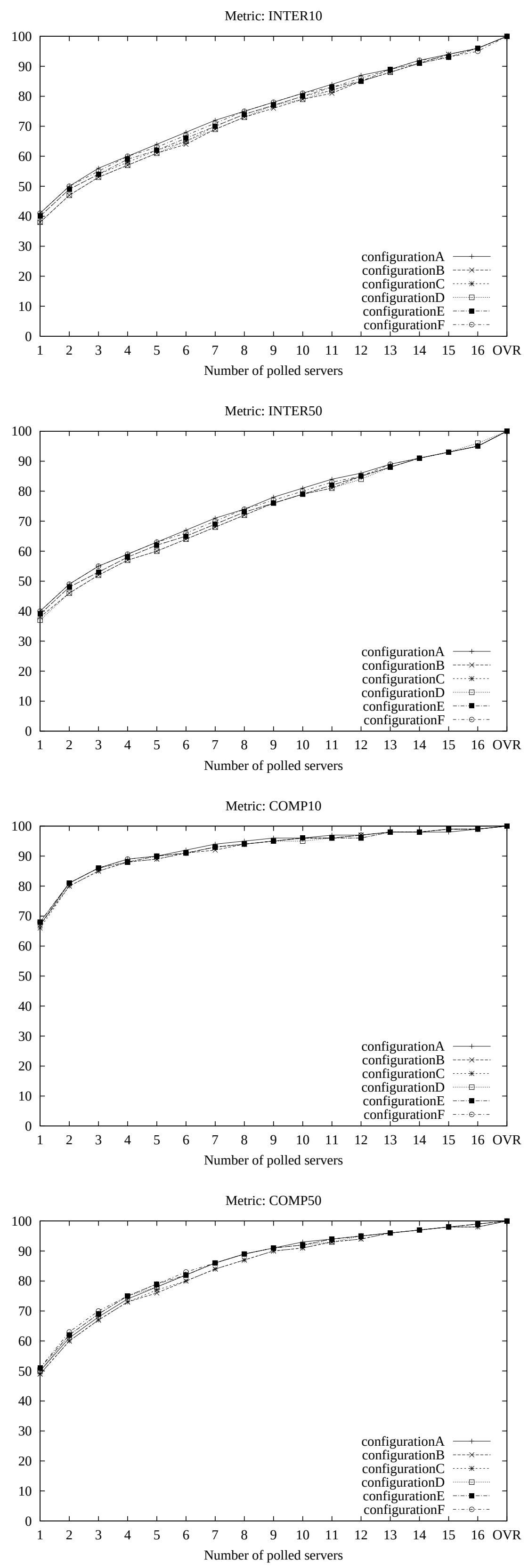

Figure 5: Intersection and competitive similarity at 10 and 50 with different configurations.

work exploits the features of our PCAP collection selection algorithm to perform an effective assignment, with a very limited computing cost (the cost of a query). The updated partitions offer a result quality similar to that reached if we reassign all documents from scratch, a more costly update procedure.

All these results were measured with exhaustive experiments conducted on a base of 6 million documents, with 190,000 queries for training and a test log of about 600,000 queries.

By using the PCAP collection selection function, we can choose the best cluster for each new document. The proposed strategy is effective in keeping the partitions reasonably balanced (with the smallest partition about half the size of the biggest one) and in maintaining the same result quality we would reach if the documents were fully reassigned using co-clustering. This simple strategy can be very effective in a real implementation because of its very limited cost.

We plan to verify if this strategy can be also used to assign silent documents, *i.e.* all the documents that are never recalled during training. While we proved that silent documents are not contributing to a big fraction of results [Puppin et al. 2006], we could use the PCAP assignment strategy to reduce the size of the supplemental index. The fact that, after co-clustering, we have coherent document and query clusters is helpful to assign documents that we cannot assign properly during the training.

This paper is adding another contribution in support of our idea of using collection selection to reduce the computing load of a search engine, the number of machines needed to serve a given number of users, the cost of buying, installing and powering them, the number of system administrators needed to keep them humming quietly. So, our dear reader, the last word is yours: you can say we are trying to save the world, or to have a bunch of innocent system administrators fired! ☺

7. REFERENCES

Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., and Raghavan, S. 2001. Searching the web. *ACM Trans. Inter. Tech. 1,* 1, 2–43.

Badue, C. S., Baeza-Yates, R., Ribeiro-Neto, B., Ziviani, A., and Ziviani, N. 2007. Analyzing imbalance among homogeneous index servers in a web search system. *Inf. Process. Manage. 43,* 3, 592–608.

Baeza-Yates, R., Castillo, C., Junqueira, F., Plachouras, V., and Silvestri, F. 2007. Challenges in distributed information retrieval (invited paper). In *ICDE*.

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

Chowdhury, A. and Pass, G. 2003. Operational requirements for scalable search systems. In *CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management*. ACM Press, New York, NY, USA, 435–442.

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

Frieder, O., Grossman, D., Chowdhury, A., and Frieder, G. 2000. Efficiency considerations in very large information retrieval servers. *Journal of Digital Information 1,* 5 (April).

Heaps, H. S. 1978. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., Orlando, FL, USA.

Lester, N., Zobel, J., and Williams, H. E. 2004. In-place versus re-build versus re-merge: index maintenance strategies for text retrieval systems. In *ACSC '04: Proceedings of the 27th Australasian conference on Computer science*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 15–23.

Li, M., Lee, W.-C., and Sivasubramaniam, A. 2004. Semantic small world: An overlay network for peer-to-peer search. *12th IEEE International Conference on Network Protocols (ICNP'04)*.

Lim, L., Wang, M., Padmanabhan, S., Vitter, J. S., and Agarwal, R. 2003. Dynamic maintenance of web indexes using landmarks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*. ACM Press, New York, NY, USA, 102–111.

Melink, S., Raghavan, S., Yang, B., and Garcia-Molina, H. 2001. Building a distributed full-text index for the web. *ACM Trans. Inf. Syst. 19,* 3, 217–241.

Puppin, D. and Silvestri, F. 2006. The query-vector document model. In *CIKM 2006*.

Puppin, D., Silvestri, F., and Laforenza, D. 2006. Query-driven document partitioning and collection selection. In *Infoscale 2006*.

Puppin, D., Silvestri, F., Perego, R., and Baeza-Yates, R. 2007. Load-balancing and caching for collection selection architectures. In *Infoscale 2007*.

Robertson, S. E. and Walker, S. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc., New York, NY, USA, 232–241.

Shieh, W.-Y. and Chung, C.-P. 2005. A statistics-based approach to incrementally update inverted files. *Inf. Process. Manage. 41,* 2, 275–288.

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

Tomasic, A. and Garcia-Molina, H. 1993. Performance of Inverted Indices in Distributed Text Document Retrieval Systems. In *PDIS*. 8–17.

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

OVER!