Two-dimensional distributed inverted files

Esteban Feuerstein[1], Mauricio Marin[2], Michel Mizrahi[1], Verónica Gil-Costa[2], and Ricardo Baeza-Yates[2]

[1] Departamento de Computación, FCEyN, Universidad de Buenos Aires, Argentina.
[2] Yahoo! Research Latin America, Santiago, Chile.

Abstract. Term-partitioned indexes are generally inefficient for the evaluation of conjunctive queries, as they require the communication of long posting lists. On the other side, document-partitioned indexes incur in excessive overheads as the evaluation of every query involves the participation of all the processors, therefore their scalability is not adequate for real systems. We propose to arrange a set of processors in a two-dimensional array, applying term-partitioning at row level and document-partitioning at column level. Choosing the adequate number of rows and columns given the available number of processors, together with the selection of the proper ways of partitioning the index over that topology is the subject of this paper.

1 Introduction

Inverted files [?] are used as index data structures to efficiently solve queries upon huge text collections. An inverted file is composed of a vocabulary table and a set of posting lists. The vocabulary table contains the set of relevant terms found in the text collection. Each of these terms is associated with a posting list which contains the document identifiers where the term appears in the collection along with additional data used for ranking purposes. To solve a query, it is necessary to get the set of documents associated with the query terms and then perform a ranking of these documents in order to select the top-K documents as the query answer. From the literature we can learn of a number of methods for distributing the inverted file onto P processors or computers and their respective query processing strategies [?,?,?,?,?,?,?,?]. Distributing an index consists of splitting the document collection and/or the index itself among the computers. There are different ways of doing this splitting, mainly variants of two basic dual approaches: document-based partition (a.k.a local indexes) and term-based partition (a.k.a global indexes). Variants of these two basic schemes have been proposed in [?,?,?].

The ranking of documents can be performed upon either intersection or union of posting lists. For queries requiring such intersection, the global indexes tend to be inefficient as they require sending complete posting lists among processors. To alleviate this problem, it has been proposed to concentrate together in the same processor terms that usually appear together in queries, reducing the probability of having to transfer posting lists. Different methods have been proposed in

[?,?,?]. On the other side, when using a local index, document IDs are assigned to unique processors and thereby the intersection of posting lists does not require communication. However, all processors must participate in the evaluation of each query: as the number of processors grows, the overhead associated with each query grows linearly, so the improvement in the throughput is not proportional to the number of processors. The reason is that each active query being processed is replicated P times on the whole set of P processors and, in each processor, they demand the use of hardware resources which do not come for free in terms of latency.

A natural idea to overcome the problems of these two approaches is to use a two-dimensional scheme trying to benefit from the advantages of the two extreme distributed indexes. The idea is to arrange a set of $P = R \times C$ processors as a matrix of R rows and C columns, applying term-partition at row level and document-partition at column level. In few words, the document collection is partitioned in C sub-collections, each of which is allocated to a "column" of R processors, which will hold the index of that collection in a term-based partitioning. The point for conjunctive queries is the following: for any concrete policy used to partition and group terms, the probability of co-residence of a pair of terms of a query increases as the number of processors decreases, so the communication cost tends to decrease with the number of rows of the matrix (the optimal being an arrangement of one single row and P columns, that is a normal document-partitioned system). At the same time, when the number of columns increases, so does the overhead typical of local indexes, so one can expect that there is an optimal configuration somewhere in between the extreme approaches. The proposal of this paper is, therefore, to analyze the performance of different configurations for a fixed number P of processors, ranging from P rows and one column (term partition) to one row and P columns.

2 A two-dimensional partitioning index

The processors form a two-dimensional array of R rows and C columns; in one of the dimensions (the rows) the index is seen as partitioned by terms, in the other dimension (columns) as partitioned by documents. The document collection is partitioned therefore in C sub-collections, each of which is allocated to a "column" of R processors, which will hold the index of that collection with a term-based partitioning. This two-dimensional scheme brings about different ways of evaluating a query. The one studied in this paper is to first distribute the query among the columns (the processors that contain query terms in each column) as with a local index. Then, at each column, the intersection must be resolved as in a global index by invoking the processors of the column that hold the required terms, and finally merging the results obtained at each column.

In the case $R = 1, C = P$ (document partitioned index), each processor holds the posting lists of the whole set of terms appearing in the documents assigned to it. Conversely, when the index is term partitioned ($R = P, C = 1$), the documents are considered as an indivisible package. As soon as we leave these extreme cases to consider a 2D scheme with more than one row and more than one column, the need appears of dealing simultaneously with term and document partitions. The question that arises is: once that C and R are fixed, how the two kind of partitions can be optimized? This question regards not only which technique or criteria is used to optimize each of them but, and this is a novelty specific to the two-dimensional partitioning of the index, how the two partitions are combined. For example, the partition of the terms could be done independently of how the documents will be partitioned. Or the terms could be partitioned taking into account the information of the document partition. Also we could partition first the terms and then the documents, and many other possibilities. To make things more complex, in each of those schemes one can use different algorithms for term and document partition, yielding an enormous amount of possibilities. There is a wide literature regarding how these partitions can be optimized (see for example [**?**,**?**,**?**,**?**,**?**,**?**]). The different trade-offs must be evaluated upon a baseline cost model which we develop in the following. In Section 3 we describe the particular algorithms we used to partition both terms and documents.

Basic cost The processing of a query can be decomposed in a series of operations that are executed in different processors. These are the primitive operations such as broadcast or communication, list intersection, merging, ranking, etc. Each of these operations has a cost, and their sum conforms the computation and communication cost of a query. In addition, each processor incurs in a certain overhead due to hardware use, network access and system scheduling tasks among others. The weight of these overheads in the total cost turns out to be high, so it cannot be neglected. In a local index the number of participating processors per query is much greater than in the global scheme.

In the following we will assume that a certain number q of queries are initially presented at every processor and then new queries arrive as the system delivers answers for previous queries. So, at every moment there are $q * P$ live queries in the system. In that framework, providing that a good load balance is obtained, we can assume that the whole set of P processors can work in parallel, and there will not be idle times. To simplify, from now on we will consider only two-term queries of the form $t_1 \wedge t_2$. We will use the following notation:

- $t_i(x, y)$: Expected time employed by a processor to compute the intersection of two lists of lengths x and y respectively.
- $t_m(x)$: Expected time employed by a processor to merge a set of lists of total length x.
- $t_r(x)$: Expected time employed by a processor to rank a list of length x.
- $I(x, y)$: the expected length of the intersection of two lists of length x and y.
- γ: time employed to transmit a unit of information from one processor to another.

Let ℓ be the expected length of a posting list (considering all the files of the system). We will assume that to prepare a result list of K results using a local index distributed among P processor, each processor will send to the originator of the query its best $2K/P$ postings and that the $2K$ results obtained that way are, with high probability, enough to answer the query. Was that not the case, another extra request would be generated for a subset of the processors, but we ignore that in this paper.

Local Index: The sequence of tasks performed *in parallel at each of the P processors* for a set of q queries, and their corresponding costs, can be described as follows:

Action	**Cost**
Broadcast the q queries of each processor to all other processors	$q(P-1)\gamma$
For each query two lists of expected length ℓ/P are intersected	$qPt_i(\frac{\ell}{P}, \frac{\ell}{P})$
For each of the $q * P$ queries, the resulting lists are ranked	$qPt_r(I(\frac{\ell}{P}, \frac{\ell}{P}))$
For each query, send $\frac{2k}{P}$ results to the originator of the query	$qP\frac{2k}{P}\gamma$
For each query originated at that proc., merge the P lists received	$qt_m(2K)$

Global Index: Let ℓ_{min} be the expected length of the shortest among the two posting lists of the terms of a query. Let $\alpha(X)$ be the probability of co-residence of the two terms of a query given that the terms are partitioned in X processors. With probability $(1-\alpha(P))$ the query should be distributed among two processors, so we need to broadcast the two terms to the two processors holding them, and the processor holding the shortest among the two lists send it to the other one. With probability $\alpha(P)$ the two terms are co-resident in one processor, so the query must just be sent to it. In both cases, the processing is completed by intersecting the two lists, ranking the result and sending the best K elements to the originator of the query. All this can be summarized in the following table. Recall that we are assuming that q queries are submitted to each processor, so the probabilities $\alpha(P)$ and $(1-\alpha(P))$ can be interpreted as fractions of the total number of queries.

Action	**Cost**
(Non co-residence) Send the terms to their two processors	$(1-\alpha(P))q2\gamma$
(Non co-residence) The shortest list is sent to the other processor	$(1-\alpha(P))q\gamma\ell_{min}$
(Co-residence) Send the two terms to one processor	$\alpha(P)q\gamma$
Intersect the two lists l	$qt_i(\ell, \ell)$
Rank the resulting list	$qt_r(I(\ell, \ell))$
The best K elements of the resulting list are sent to the originator	$qK\gamma$

2D Index: We will analyze this model assuming we have R rows and $C = P/R$ columns. The sequence of tasks to be developed at each processor (always assuming q queries per processor) starts with the broadcast of the q queries to each of the C columns (to a random processor at the column). The R processors of each column must then resolve a total of qP queries, so each one of them will hold expectedly $\frac{qP}{R} = qC$ queries. So this part will be executed in parallel by the C columns, and within each column by the R processors of the column, therefore we can think that the P processors are working in parallel. In each column the terms may be co-resident at the same processor (row) or not, with probabilities respectively $\alpha(R)$ and $(1-\alpha(R))$, so different tasks will be executed for the corresponding fraction of the queries. After that, always at column level, but with the C columns working in parallel, intersection and ranking of the lists ($q * C$ queries at each processor). Finally, each column (actually, the processor in the column that has computed and ranked the intersection) must send its results to the originator of the query, that will merge the results.

Action	**Cost**
Broadcast the q queries to a random processor in each column	$qC\gamma$
(Non co-residence) Send the two terms and then send the shortest list from one processor to the other one	$(1-\alpha(R))*$ $(qC2\gamma + qC\gamma\ell_{min})$
(Co-residence) Send the two terms to their processors	$\alpha(R)qC\gamma$
$q * C$ intersections of lists of expected length ℓ/C	$qCt_i(\ell/C)$
$q * C$ rankings of the lists at each processor	$qCt_r(I(\ell/C, \ell/C))$
For each of the qC queries, send $\frac{2K}{C}$ results to its originator	$qC\frac{2K}{C}\gamma$
Merge the C lists of length $2K/C$ received in each processor	$qt_m(2K)$

Overhead To compute the real cost associated with a query we have to add to the expressions developed in the previous section a fixed cost or overhead (that we will denote as β). This will be counted for every processor participating in a query. In a local index each query will have an overhead of $P * \beta$. In a global index the terms may be co-resident or not at each column, so the overhead may be seen as a random variable with expected value $(\alpha(P) + (1-\alpha(P)) * 2) * \beta$. Finally, in the general 2D case with R rows, one or two processors participate at each column so the expected value of the overhead is $C*(\alpha(R)+(1-\alpha(R))*2)*\beta$.

3 Experimental evaluation

For term partition we used a term-clustering heuristic oriented to reducing communication cost and at the same time maintaining the load balance of the system. This heuristic, based on the one used in [**?**], tries to assign to the same machine pairs of terms of high cost (a function of the relative frequency and length of the shortest posting lists of its terms). We will refer to this heuristic as TCH. The basic heuristic that we used for document clustering tries to group similar documents (cosine measure) and assign them to the same processor. It starts with a certain number of documents that are chosen initially as cluster centers. These cluster centers are selected so that they are sufficiently different from each other. Then we insert into each cluster the documents that are closer to each cluster center. Finally, the clusters are assigned to the different machines in a round-robin fashion [**?**,**?**]. We will refer to this heuristic as DCH. For document partitioning we also consider a simple Random partition (DRH). The first two-dimensional heuristics we considered were to partition terms and documents independently, using TCH for terms and either DRH or DCH for the documents. We will refer to these heuristics as 1.a and 1.b respectively.

Another family of heuristics consists in partitioning first the documents, using either DRH or DCH, and then the terms using TCH separately for each column, taking into account the documents that were assigned to each column (heuristics 2.a and 2.b respectively). Also by first partitioning documents we could use the information of that partition to produce one single partition for the terms to be used across all the columns. These heuristics will be referred to as 3.a (with DRH) and 3.b (with DCH).

A different approach may be taken if we first partition the terms and then the documents. Given as input an initial partition of the terms, the heuristic 4 tries to distribute the documents among the columns so as to minimize the communication cost. We consider only pairs of non co-resident terms (as co-resident pairs will not require further communication). The intuition behind the heuristic is to try to minimize in two ways the lengths of the posting lists that must be transfered: (a) separating in different columns documents that *are not* part of the intersection of popular pairs, and (b) minimizing the length of the shortest posting list at each column by increasing the variance of the lengths of the lists. The (last) heuristic 5 constructs the partitions of terms and documents simultaneously, considering pairs of queries one by one, in decreasing order of cost. For each query, it decides whether to consider it to group together its terms, or to separate the documents of their posting lists.

The final expression for the cost *of a single query* will be obtained by considering the computation and communication costs plus the overhead incurred by every participating processor. For that, we need to adapt the values given in Section 2 to an individual query instead of a set of q queries, getting a per-query cost of:

$$\begin{aligned} C\gamma + (1-\alpha(R))(C2\gamma + C\ell_{min}\gamma + 2C\beta) + \alpha(R)(C\gamma + C\beta) + \\ Ct_i(\ell/C) + Ct_r(I(\ell/C,\ell/C)) + C\frac{2K}{C}\gamma + t_m(2K) \end{aligned} \qquad (1)$$

This formula is valid for the case in which the term partition is uniform across all the columns (i.e. the two terms of a query are assigned to the same row at each column), and therefore are co-resident or not uniformly in all the columns.

We did our experiments on two inputs: Collection 1 is a sample of the Chilean web with $\approx 160K$ documents, Collection 2 contains a subset of $\approx 2M$ documents of a 1.5TB sample of the UK's web and $\approx 250K$ queries taken from a one-year log of a major search engine's site. We simulated and measured the performance of every heuristic with different configurations on $P = 256$ processors. The number of rows ranged from 1 (local indexing) to 256(global-indexing) using successive powers of two.

For the simulation we defined particular costs for the different primitive functions, based on benchmarking runs we did on the same collections. The values are expressed relative to a base-line in terms of ranking time defined as $t_r(x) = x$. Intersection and merge operations require in average $t_i(x, y) =$

$\min(x \log(y), x + y)/4$ and $t_m(x) = x/4$ respectively. The values for β and γ where chosen to achieve proper agreement with what we have observed using two actual implementations of document- and term-partitioned inverted files for disjunctive queries. We run experiments on the two indexes, in which the pure global index resulted on average 20% more efficient than the pure local one, so the values for β and γ where chosen so as to satisfy that relation.

The graphics in figure 1 summarizes the results of our simulation. It shows total costs (processing+communication+overhead) as a function of the number of columns, for the two collections. All the costs were normalized by dividing them by the maximum cost, that occurs for all the heuristics when the number of columns is 256 (i.e. when the 2D index becomes a simple local index). We observe that an important improvement in the cost is achieved by arranging the 256 processors in a two-dimensional array, of 8×32 or 4×64, for all heuristics. Therefore, the main claim that an improvement can be obtained with a 2D index against the classical local and global indexes is verified.

It may be observed that there is not a big difference in the performances of the heuristics, although heuristics 4 and 5 behave consistently better than the others in almost all configurations (the latter being a bit better in general). Note that heuristics 4 and 5 cannot be applied for simple local and global indexes. These seem to be the only heuristics that take advantage of the two-dimensional structure and the possibility of combining clustering techniques for terms and documents. The results shown in the figure were computed for particular values of the parameters β and γ. The difference between the best and worst configurations is of more than 20%.

4 Conclusions and further work

The preliminary results obtained in our simulations are a positive signal towards the continuation of our study in that direction. An immediate task we have to focus on is the realization of further and deeper experiments, with real executions in real environments, with larger document collections and query logs. Those experiments should include the usage of different total number P of processors.

An interesting subject of further research regards the possibility of dynamically reconfiguring the arrangement of the processors to adapt to different types of queries, and also the use of non rectangular arrangements (rows or columns of different length). Finally, we plan to analyze how do different ranking policies at row and column level may affect the performance of the system.

References

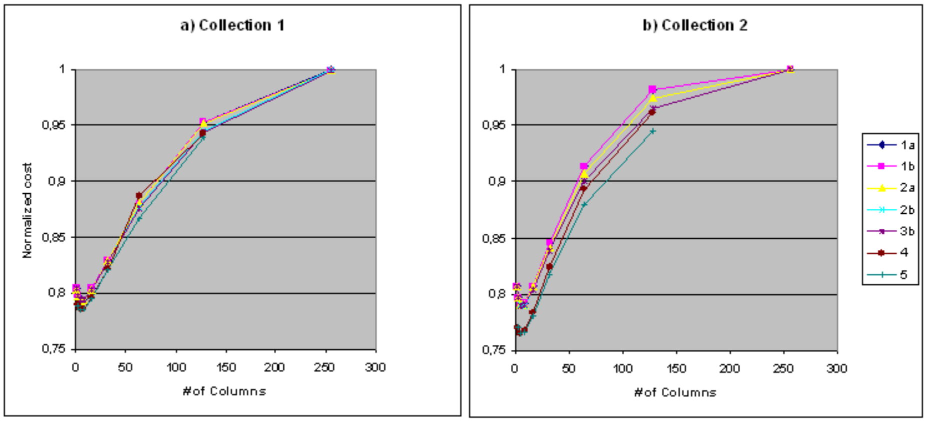

Fig. 1. Normalized costs as a function of number of columns, for different heuristics