

# Query Performance for Tightly Coupled Distributed Digital Libraries

Berthier A. Ribeiro-Neto\*  
Ramurti A. Barbosa†  
Computer Science Department  
Federal University of Minas Gerais  
Brazil  
{berthier,ramurti}@dcc.ufmg.br

## ABSTRACT

We consider a digital library distributed in a tightly coupled environment. The library is indexed by inverted files and the vector space model is used as ranking strategy. Using a simple analytical model coupled with a small simulator, we study how query performance is affected by the index organization, the network speed, and the disks transfer rate. Our results, which are based on the Tipster/Trec3 collection, indicate that a global index organization might outperform a local index organization.

**KEYWORDS:** digital library, distributed, query performance

## Introduction

The traditional model of text retrieval is that of a collection of documents indexed by keywords. In this model, the user specifies his information need by providing sets of keywords and the information system retrieves the documents which best approximate the user query. Further, the information system might attempt to rank the retrieved documents using some measure of relevance. Such model is simple and elegant and has prevailed for many years. In the last five years, however, the popularization of the Web (i.e., World Wide Web) has changed the scenario quite a bit which has led to the coining of the term *digital library*. While it is not clear what exactly the term refers to, it does carry a semantic load implying that it is more than a conventional information retrieval (IR) system. In fact, current digital libraries are becoming much more complex systems which include, besides text search, functionalities related to hypertext, multimedia, the Web (i.e., the World Wide Web), and highly interactive interfaces.

Despite all this additional complexity, the favorite form of searching for information continues to be (it seems) through

the specification of keywords. The main reason seems to be the convenience of the model to the users, the simplicity of its implementation, and its good performance with large collections. To achieve good searching performance with large collections, however, it is necessary to have all the documents (in the collection) stored locally in a central machine or distributed in a tightly coupled environment. In fact, all Web search engines replicate the Web pages (or most of them) locally, index them, and do keyword based searching in this local digital library.

In the setting of a large local digital library, three possible hardware environments for operation are a central (large) machine, a parallel machine, or a network of workstations (i.e., a set of workstations tightly connected by fast switching equipment). In this work, we focus our attention on the operation of a digital library distributed in a network of workstations (or machines) *tightly coupled* (i.e., with communication speeds of 100M bits per second or more). Particularly, we are mostly interested in query performance issues.

For efficient query processing, specialized indexing techniques have to be used with large digital libraries. A number of distinct indexing techniques for text retrieval exist in the literature and have been implemented under different scenarios. Some examples are suffix arrays, inverted files, and signature files [5]. Each of them have their own strong and weak points. However, due to its simplicity and good performance, *inverted files* have been traditionally the most popular indexing technique used along the years. Therefore, in this work, we consider that the digital library is indexed using inverted files.

Given an inverted file (or a set of them) for all the documents in the distributed library, it is necessary to consider how to store such index. Since there are several machines in the network, it is reasonable to distribute the index among them. Two basic and distinct index distributions have been characterized in [18]. In the first one, each machine generates an inverted file for its local documents and stores this index locally. In the second one, a global inverted file (for all the documents in the library) is generated and distributed among the various machines. In this work, we study the problem of processing a batch of queries with these two index organizations in a distributed digital library.

In our distributed digital library, a query is specified as a set of

---

\*Partially supported by Brazilian CNPQ grant 300188/95-1.

†Partially supported by a Brazilian CNPQ scholarship.

keywords. Given the query, the library search engine looks for (using an inverted file as index) the documents which contain query terms — a search which might involve several workstations. These documents are then ranked using the vector space model [16] and the documents in the top of this ranking are returned.

We develop a simple analytical model to predict query performance in our distributed digital library. We first show that such model, when coupled with a small simulator, yields estimates which are quite accurate. Following, we evaluate the execution time for processing a batch of 50 Trec3 queries [7] under various scenarios. Our results indicate that a global index organization might become quite advantageous (with respect to a local index organization) in the presence of fast communication channels.

The paper is organized as follows. We first briefly cover related work. Following, we present the network architecture and characterize our concept of a distributed digital library. Afterwards, we detail the query processing strategy for this library and develop a simple analytical model for predicting query performance. Our results and conclusions follow.

### Related Work

The work in [3] simulates a collection of servers in a local area network but does not focus on the physical organization of the index as we do. The work in multiprocessor information retrieval systems [1, 11] is related to our work in the sense that the index organization also strongly affects query performance. In [17], a parallel SIMD algorithm for document retrieval is discussed with no distinction between local and global indexes. In [10, 12], various data partitioning schemes are studied which is a work complementary to ours. In [4], a distributed architecture which uses a connection server to associate clients to servers is simulated.

The work which is closest to ours is that in [18]. However, in that work, the authors consider that the processing of queries is done using the boolean model with no consideration of a ranking strategy. Therefore, their conclusions are not directly applicable to modern digital libraries which depend fundamentally on a ranking strategy for retrieving relevant information. We depart from their work in four main directions. First, instead of considering the boolean model, we adopt the vector space model [16] which is far more popular nowadays. Second, instead of considering conjunctive queries, we consider disjunctive queries as required by the vector space model (which implies that we do not have the reduction in the answer set provided by conjunctive operations). Third, instead of modeling the document collection and the queries, we base our results on the Tipster/Trec3 collection [7] and its set of real queries (which implies that we have to worry about recall and precision figures). Fourth, instead of relying entirely on a simulation, we develop a simple analytical model which is fairly accurate. Our study identifies tradeoffs which are not present in their work but also shows that some of their results remain valid in the scenario of our distributed digital library. For instance, in the presence of a fast communication channel, we show that a global index might become quite advantageous.

### Network Architecture

The environment we select for operation of our distributed digital library is that of a network of workstations connected by fast switching technology [2]. A network of workstations is an attractive alternative nowadays due to the following reasons. First, the emergent fast switching technology (such as ATM) provides very fast message exchanges among the various machines and consequently less parallelism overhead. Second, a network of workstations provides computing power comparable to that of a typical parallel machine but is more cost effective [2]. Third, these workstations are already available elsewhere and most of them are completely idle most of the time [2].

The distributed memory in our network has a shared-nothing organization. This implies that all communication is through messages originated directly by the processes in our application (i.e., there is no interference from operating system memory control processes such as those present in shared memory monitors). This is important to ensure that exchanging data in our system is as fast as the communicating channels (and the associated overheads due to the minimum indispensable communication protocols) allow.

Figure 1 illustrates the network architecture and distributed (shared-nothing) memory which compose the distributed environment for our digital library. Notice that all the disks are local and can be accessed directly (by the local machine) without going through the network.

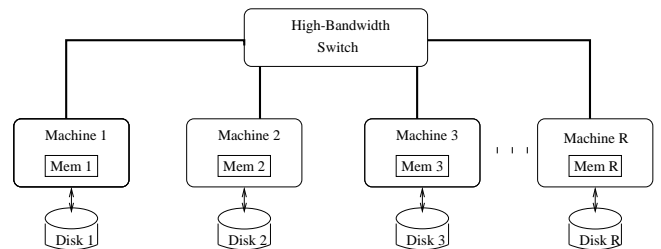


Figure 1: Network architecture and distributed (shared-nothing) memory.

### Distributed Digital Library

In this section we describe our conception of a distributed digital library which operates in the network environment depicted above. Our discussion covers the distribution of the documents across the network, the choice of inverted lists as indexing structure, the distribution of the index across the network, the vector space model ranking strategy, and the processing of the queries.

*Distribution of Documents in the Network* The main objective of this study is not to provide numerical estimates of query execution time for various scenarios but rather to identify the main issues and the main tradeoffs involved with the processing of a batch of queries in our distributed digital library. Thus, for clarity, we focus our attention on identifying such tradeoffs and consider only the case in which the documents are evenly distributed across the network.

Let  $r$  be the number of machines in the network and  $n$  be the size (in bytes) of the whole collection. Define,

$$b = \frac{n}{r} \quad (1)$$

Then, considering that the documents are evenly distributed across the network, each machine holds (in its local disk) a subcollection whose size (in bytes) is roughly given by  $b$ .

*Indexing with Inverted Files* Inverted files are useful because, although the size of the index is proportional to the size of the text, their searching strategy is based mostly on the vocabulary (i.e., the set of distinct words in the text) which usually fits in main memory<sup>1</sup>. Further, inverted files are simple to update and perform well when the pattern to be searched for is formed by conjunctions and disjunctions of simple words which is probably the most common type of query in information retrieval systems. Thus, we consider that our distributed digital library uses inverted files as its only indexing structure.

An *inverted file* is an indexing structure composed of: (a) a list of all distinct words in the text which is usually referred to as the *vocabulary* and (b) for each word  $w$  in the vocabulary, an *inverted list* of documents in which the word  $w$  occurs. Additionally, the vocabulary is sorted in lexicographical order. Since we adopt the vector space model (as discussed later on) to rank the documents in the query answer set, the inverted lists need to keep information on term frequencies. This is accomplished by adding four bytes to each document entry in the inverted lists. For a document  $d$  and a word  $w$ , these four bytes hold the normalized weight associated by the vector space model to the pair  $(d, w)$ .

With large texts, some restrictions are imposed on the inverted file to keep it smaller [8]. Examples of these restrictions are: (a) filtering of text characters and separators, (b) use of a controlled vocabulary in which not all words in the text are indexed, and (c) exclusion from the vocabulary of *stop words* such as articles and prepositions. The main disadvantage of these restrictions is that words excluded from the index but present in the text are not searchable.

*Index Organization* In our distributed library, each machine holds a subcollection whose size (in bytes) is roughly  $b$ . This implies that, for each subcollection, the corresponding inverted file has size which is  $O(b)$ . Thus, for  $r$  machines, the size of the index for the whole library is given by  $r \times c_1 \times b$  where  $c_1$  is a proportionality constant. We consider two basic organizations for this library index. The first one considers only local inverted files while the second one considers a global inverted file for the whole library [18]. We discuss both of these organizations in the immediately following.

One possible organization for the library index is to have each machine with its own local inverted file. This organization is called disk index in [18] but we prefer to call it *local index organization* which seems more direct. In the local index organization, generating and maintaining the indexes

<sup>1</sup>In a text of size  $n$  it is reasonable to expect a vocabulary of size proportional to  $\sqrt{n}$  [9].

is simple because everything can be done locally without interaction among the machines. However, processing a given query requires sending the whole query to every machine in the network. Furthermore, each machine has to process the whole query. As we show later, this might result in a considerable disadvantage.

Since the indexes are local to each machine, global information on the occurrence of keywords in the collection is missing. Without this information, the estimates for the *inverse document frequency* (*idf*) weights (associated to each term by the vector space model [16]) are slack and, as a result, the global ranking generated suffers from a drop in precision figures. To exemplify, consider the collection of documents in disk 1 of the Tipster/Trec3 collection and the respective set of queries numbered from 101 to 150 [7]. For this collection, figure 2 illustrates the drop in average precision as the number  $r$  of machines increases. To avoid this drop in

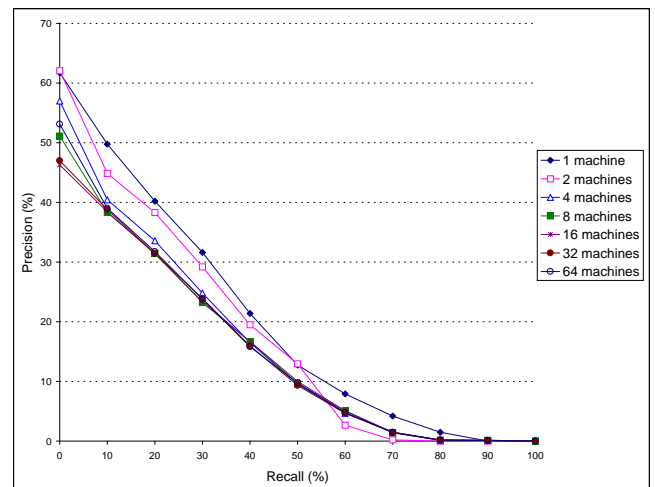


Figure 2: Recall versus precision figures for local index organization without global *idf* weights (50 Trec3 queries).

average precision, it is necessary to provide each machine in the local index organization with access to global information on keywords distribution. This can be accomplished by computing the inverse document frequencies for all the words in the vocabulary and distributing this information to all the machines. Fortunately, the vocabulary is small enough such that it easily fits in main memory. Thus, in the remaining of this work we consider that, for the local index organization, each machine keeps a copy of the global vocabulary (with globally computed weights) in its local main memory. By doing so, we guarantee that the answer sets generated by our distributed digital library are as *precise* as those generated had the library been resident in a single, large, central machine.

The other clear alternative to the local index organization is to have a global inverted file for the whole library. To the best of our knowledge, this is the approach taken by most search engines in the Web. These engines seem to maintain a global library (composed of copies of most of the Web documents) for which a global index is periodically recomputed. As it

seems, this global index is normally maintained in a single, large, central machine. Here, we consider the situation in which this global index is distributed among a set of machines in a network tightly coupled.

While there are many possibilities to distribute the index (for instance, the distribution might be based on a criteria of load balancing), for simplicity, we consider that the global index is distributed among the machines in lexicographical order such that each machine holds roughly an equal portion of the whole index. According to this strategy, machine 1 might end up holding the global inverted lists for all the keywords which start with the letters A, B, or C, machine 2 might end up holding the global inverted list for all the keywords which start with the letters D, E, F, or G, and so on. The important issue for us here is that each machine holds a portion of the global index of size roughly equal to  $b$ . In this way, we can expect that the comparison between this organization and the local index organization is a fair one.

Distributing the global index among the various machines is an organization called *system index* in [18]. Here, however, we prefer to call it simply *global index organization* because it seems a more direct reference.

**Ranking with the Vector Space Model** In this work, we adopt the vector space model as the ranking strategy for our digital library because it is one of the most widely used models in information retrieval systems. The main reason is its simplicity and its good retrieval performance with general collections [6, 15, 16].

In the vector model, documents and user queries are represented as vectors of keywords. Let  $\vec{d} = (k_1, \dots, k_t)$  be the vector representing the document  $d$  and  $\vec{q} = (k'_1, \dots, k'_t)$  be the vector representing the user query  $q$ . To improve precision and recall levels, the vector model associates weights to document and query indexes. Let  $w_{i,d}$  be the weight associated to the keyword  $k_i$  of document  $d$  and let  $w_{j,q}$  be the weight associated to the keyword  $k_j$  of query  $q$ . These weights are usually specified as a variation of *tf-idf* factors [15]. The similarity of a document  $d$  with respect to a user query  $q$  is measured as the cosine of the angle between the weighted query and document vectors. This *cosine similarity ranking formula* can be written as

$$\text{sim}(\vec{d}, \vec{q}) = \frac{\vec{d} \bullet \vec{q}}{|\vec{d}| \times |\vec{q}|} \quad (2)$$

$$= \frac{\sum_{i=1}^t w_{i,d} \times w_{i,q}}{\sqrt{\sum_{i=1}^t w_{i,d}^2} \times \sqrt{\sum_{i=1}^t w_{i,q}^2}} \quad (3)$$

This is the ranking strategy adopted in our distributed digital library.

### Query Processing

We assume that there is a *central broker* machine to which all the queries are first directed. This broker inserts the query requests in a queue and process them from there. In this study, we do not evaluate the impact of the query arrival pattern on the system performance. Instead, we assume that

there are always enough queries to fill a minimum size query processing queue. Since our experiments use queries from the Trec3/Tipster collection, we assume that this query processing queue has a size equal to 50. Given this batch of 50 queries, the broker processes them in slightly different ways for the local and global index organizations as we now discuss.

In the local index organization, the broker takes a query out of the queue and sends this query to all the machines in the network. Each machine then processes the whole query locally, obtains the set of documents related to that query, ranks them using the vector space model, selects a certain number of documents from the top of the ranking, and returns them to the broker as the local answer set. The central broker then collects the  $r$  sorted local answer sets and combines them (through a merging sort procedure) into a global (and final) ranked set of documents.

By selecting a set of documents from the top of the ranking, each machine reduces the amount of data which has to be sent (to the broker) through the network. However, such reduction in network traffic must not affect the precision of the global answer set. This can be assured through the adoption of a simple strategy as follows. Consider, for instance, that global precision is evaluated through the first  $f_1$  documents in the top of the ranking as suggested for early Trec experiments (which adopted a value for  $f_1$  equal to 200) [13, 14]. Then, in the local index organization, each individual machine needs to send only its  $f_1$  top documents to the broker to guarantee that overall precision is not diminished. This is so because any document among the top  $f_1$  in the global ranking has to be among the top  $f_1$  in one of the local rankings. Thus, for the local index organization, we consider that each machine sends to the central broker only its top  $f_1$  documents where  $f_1$  is the number of documents the broker effectively returns as the global answer set.

In the global index organization, the processing of a query is different and works as follows. The central broker takes a query out of the queue and first determines which machines hold inverted lists relative to the query terms. Notice that, in this case, not all machines might be involved. For instance, not all machines are involved with the query processing whenever there are more machines in the network than query terms. Further, it might happen that many query terms are mapped to a single machine releasing neighbor machines. The situation here is quite distinct from that with the local index organization because the inverted lists in each machine hold information about all the documents in the library. Thus, for instance, if a single machine happens to hold (the inverted lists for) all the terms of a given query then that machine is able to process that query by itself without need to cooperate with any other machine. As a result, more than one query might be processed simultaneously. This is important because it might allow high concurrency which is not present with the local index organization (which, in turn, always provides high parallelism because all the machines are devoted to the execution of a single query).

Once the central broker has determined which machines hold inverted lists associated to the query terms, it breaks the

query in subqueries and sends them to the respective machines. Each subquery is composed only by the terms which are present in the machine it is sent to. Once a machine has

Number of machines	Precision versus recall										
	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
1	61.71	49.75	40.20	31.62	21.38	12.73	7.89	4.18	1.50	0.00	0.00
2	61.86	48.70	39.40	31.84	20.67	12.43	7.51	3.05	0.16	0.00	0.00
4	61.38	49.33	38.94	30.50	18.68	11.69	7.31	2.63	0.17	0.00	0.00
8	60.91	48.92	39.04	30.95	21.91	11.87	7.09	3.66	0.16	0.00	0.00
16	61.69	49.15	40.25	32.68	22.16	12.78	7.05	3.72	0.16	0.00	0.00
32	62.57	49.25	40.20	32.61	22.13	12.92	7.18	3.73	0.16	0.00	0.00
64	62.57	49.19	40.25	32.63	22.12	12.92	7.59	3.73	0.16	0.00	0.00

Figure 3: Recall versus precision figures for global index organization (50 Trec3 queries).

received its subquery, it searches for the related documents, ranks the resultant document set, selects a number of documents from the top of the rank, and sends these to the central broker. The central broker then merges the local answer sets into a global answer set and recomputes the ranking. Notice that the broker cannot use the local rankings generated by individual machines because such rankings are based only in the partial information present in the subqueries.

There are two important differences between the query processing for the global and local index organizations. First, in the global index organization, the inverted lists are larger (because they contain documents from all the library). Second, in the global index organization, the local ranking considers only partial information because there is no information on the query terms which reside on the other machines. As a result, the local ranking is less precise which implies that more documents have to be sent to the central broker (i.e., the *cutting* strategy has to be more lenient). Both of these two effects are prejudicial to the global index organization. However, there are also counter effects which favor the global index organization and, as we shall see, might become determinant.

Regarding the cutting strategy for the global index organization, we did a few experiments which showed that the cutoff factor depends on the number  $r$  of machines in the network. Thus, we adopted a conservative cutoff factor given by  $r \times f_2$  where  $f_2 = 5 \times f_1$ . Using such cutoff factor, we observed no significant variation in global precision as illustrated in figure 3.

### Analytical Model

In this section, we present a simplified analytical model for the querying process in our distributed digital library. Our model covers both the global index and local index organiza-

tions. Despite its simplicity, the model captures well the key variables which determine query processing performance for our digital library.

**Basic Variables and Critical Parameters** In what follows, we provide notation for basic variables and critical parameters. These parameters were identified through a series of experiments.

$r$ : number of machines or processors

$\vec{q}_i$ : vector of keywords which compose the  $i$ th query

$q_i$ : number of keywords in query  $\vec{q}_i$

$\vec{q}_{i,j}$ : vector composed of those keywords extracted (by the broker) from query  $\vec{q}_i$  and sent to machine  $j$

$q_{i,j}$ : number of keywords in query  $\vec{q}_{i,j}$

$l_k$ : size (in bytes) of global inverted list for keyword  $k$

$l_{k,j}$ : size (in bytes) of local inverted list for keyword  $k$  at machine  $j$

$c_{i,j}$ : number of comparisons and swaps during ranking of documents for query  $\vec{q}_i$  at machine  $j$  (local index)

$cs_{i,j}$ : number of comparisons and swaps during ranking of documents for subquery  $\vec{q}_{i,j}$  at machine  $j$  (global index)

$c_{i,b}$ : number of comparisons and swaps during ranking of documents for query  $\vec{q}_i$  at the central broker

$f_1$ : number of documents from the top of the ranking which are returned as the local answer set with the local index organization

$f_2$ : proportionality constant which affects the number of documents from the top of the ranking which are returned as the local answer set with the global index organization

$tc$ : average time (per byte) to compare two terms and swap them

$ts$ : average seek time for a single disk

$tr$ : average time to read a byte from disk and do its processing (excludes seek time)

$tt$ : average time to transfer a byte from one machine to another

$t_i$ : total time (in seconds) to complete processing of query  $q_i$

$t_{i,j}$ : total time (in seconds) to complete processing of subquery  $\vec{q}_{i,j}$  at machine  $j$

In the local index organization, the whole query  $\vec{q}_i$  is passed to all the machines in the network. This implies that each machine has to process a query whose length is given by  $q_i$ . In the global index organization, only a part of the user query is sent to each machine (in fact, a given machine might receive no query terms). This implies that each machine  $j$  has to process only a subquery  $\vec{q}_{i,j}$  whose length is given by  $q_{i,j}$ .

Once a machine  $j$  receives a query  $\vec{q}_i$  (or a subquery  $\vec{q}_{i,j}$ ) for processing, it reads (from disk) the inverted lists relative to the terms in the query  $\vec{q}_i$  and merges such lists to form the local document answer set. Considering that the inverted list for term  $k$  has size given by  $l_{k,j}$ , the number of bytes read from disk is given by  $\sum_{k \in \vec{q}_i} l_{k,j}$ .

The parameters  $c_{i,j}$ ,  $cs_{i,j}$ , and  $c_{i,b}$  count the number of comparisons and swaps for the ranking task at the various machines. The parameters  $f_1$  and  $f_2$  are used to determine the

cutoff limits for the local answer sets. The remaining parameters are related to time measures.

**Query Processing** During the processing of a query, we distinguish five main steps: (S1) seeking disks, (S2) reading inverted lists from disk and processing weights according to the vector model, (S3) local ranking of retrieved documents, (S4) transferring of ranked documents to the central broker, and (S5) final ranking at the central broker. Steps (S1) to (S4) have to be done at each machine involved in the query processing while step (S5) involves only the central broker. Further, the execution of the five steps above is normally overlapped and interleaved.

In the local index organization, each machine in the network must execute the whole query locally. This implies that the processing of a query  $\vec{q}_i$  lasts for a time  $t_i$  which is given roughly by:

$$\begin{aligned} t_{i,j} &= q_i \times ts + & (S1) \\ &\sum_{k \in \vec{q}_i} l_{k,j} \times tr + & (S2) \\ &c_{i,j} \times tc + & (S3) \\ &f_1 \times tt & (S4) \end{aligned} \quad (4)$$

$$t_i = \max(t_{i,j}) + c_{i,b} \times t_c \quad (5)$$

Notice that the parameters  $c_{i,j}$  and  $c_{i,b}$  have to be estimated accurately. This is done by running a real execution of the ranking algorithm (i.e., a sorting) and counting the number of comparisons and swaps. Furthermore, to reduce the execution time of step (S2), it is not necessary to read all the  $l_{k,j}$  bytes in the inverted list for the keyword  $k$ . By adopting a filtering technique (for the vector space model) based on the work in [14], the number of bytes read from disk can be considerably reduced (with no effect on global precision). This means that the values for  $l_{k,j}$  need also to be measured by running a real execution of the disk access pattern.

Consider for a moment that only a single machine is available. In this situation, there is no distinction between a client machine and the central broker (i.e., steps (S3) and (S4) do not need to be executed). Using the disk 1 of the Tipster/Trec3 collection, we ran queries numbered from 101 to 150 on a single PC. This PC is a Pentium 166MHz (with a memory of 64M bytes) running the Linux operating system. We also evaluated the execution time of these same 50 queries using our analytical model above. The results are as shown in figure 4. As it can be seen, the agreement between the real and the predicted execution times is rather good in this case.

Consider now the same execution above but with several machines in the network. In this case, the two components of our model which behave differently are the steps (S3) and (S4). Regarding the step (S3), we know that our prediction is fairly accurate because it worked for the step (S5) which is entirely analogous. In fact, we timed the sorting procedure several times and confirmed that our predictions for the steps (S3) and (S5) are accurate ones. Regarding step (S4), we know that it involves solely transmission of data across the network. If the time  $tt$  is estimated with care in order to take into consideration the overheads (due to the operating system and the communication protocols) which affect the

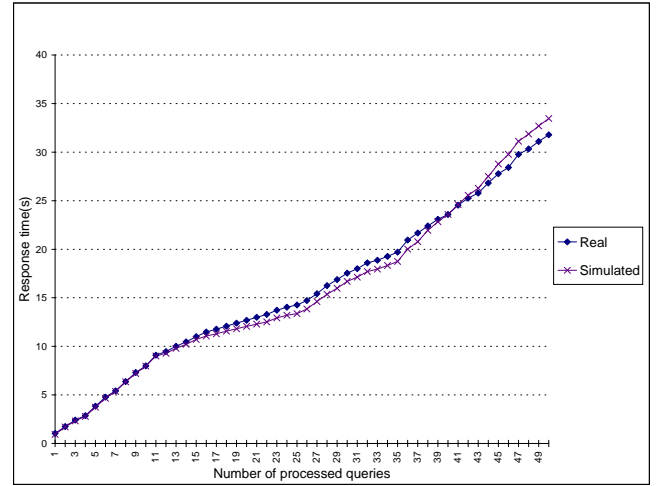


Figure 4: Comparison between real and estimated execution times for 50 Trec3 queries. Parameters are as follows:  $ts = 10ms$ ,  $tr = 0.464\mu s$ ,  $tc = 0.255\mu s$ ,  $tt = 0.1\mu s$ .

transmission of data (i.e.,  $tt$  must reflect the *net* available bandwidth), the prediction for step (S5) is also fairly accurate. The effect which remains to be captured is the interference among the various queries in our distributed environment. Such effect is taken into account as follows.

In the distributed processing of a query  $\vec{q}_i$ , some machines finish the execution of the step (S4) before others. These machines can then start immediately the execution of a following query (i.e., they do not need to wait for the conclusion of the step (S5) for query  $\vec{q}_i$ ). We capture such behavior through the use of a small simulator written in C++. This simulator replicates the execution of steps (S1) through (S4) in each machine and the execution of step (S5) in the broker for a batch of queries. The execution times of each step are estimated according to equations 4 and 5. By doing so, we are able to predict with fair accuracy the execution time for a batch of queries running in the distributed environment of our digital library. Let us now turn our attention to the query processing with the global index organization.

In the global index organization, the broker first determines which machines contain inverted lists relative to the query terms. The reason is that only those machines are involved in the query processing. Once this is done, subqueries are dispatched to the machines involved. The time  $t_i$  to process a query  $\vec{q}_i$  can then be modeled as follows.

$$\begin{aligned} t_{i,j} &= q_{i,j} \times ts + & (S1) \\ &\sum_{k \in \vec{q}_{i,j}} l_k \times tr + & (S2) \\ &cs_{i,j} \times tc + & (S3) \\ &r \times f_2 \times tt & (S4) \end{aligned} \quad (6)$$

$$t_i = \max(t_{i,j}) + c_{i,b} \times t_c \quad (7)$$

Notice that, in this case, the ranking in step (S3) is done considering the terms in the subquery  $\vec{q}_{i,j}$  only. Thus, such

ranking is not as accurate as the local ranking done with the local index organization. This implies that a larger number of documents has to be transmitted to the central broker to ensure that the precision of the global answer set is not diminished. It is due to this constraint that the step (S4) above considers that  $r \times f_2$  documents are transmitted to the central broker which ensures that global precision figures are not affected.

As done for the local index organization, we capture the interference among the various queries using the simulator we wrote in C++. Furthermore, we again adopt a filtering technique based on the work in [14] to reduce the execution time of the step (S2).

## Results

In this section, we use our analytical model coupled with our small simulator to investigate the system performance for processing a batch of queries in our distributed digital library. The two indexing schemes considered are the local index and the global index organizations as discussed before. All our estimates are based on the documents in the disk 1 of the Tipster/Trec3 collection [7]. The queries used were those numbered from 101 to 150 in the Trec3 proceedings. Unless we explicitly say otherwise, the time parameters used in our analysis (which are valid in our test machine) are as follows.

$$\begin{aligned}
 ts &= 10 \times 10^{-3} \text{ seconds (seek time)} \\
 tr &= 4.64 \times 10^{-7} \text{ seconds per byte (disk read + processing)} \\
 tc &= 2.55 \times 10^{-7} \text{ seconds per byte (comparison at cpu)} \\
 tt &= 1 \times 10^{-7} \text{ seconds per byte (network transfer time)}
 \end{aligned}$$

In figure 5 we show, for the local index organization, how the estimated time to process the 50 test queries varies with the number of machines in the network. We observe that

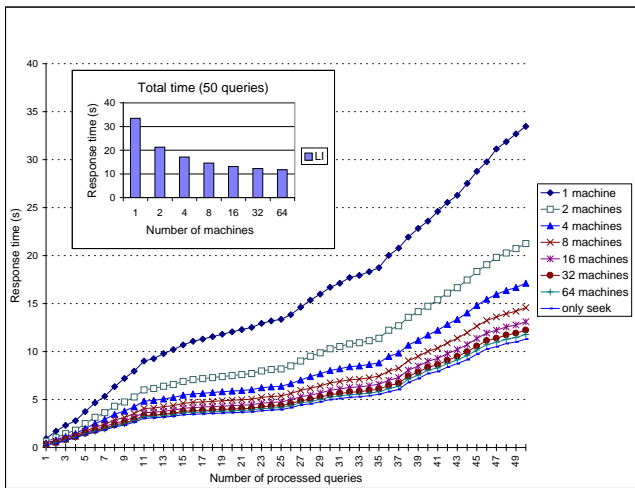


Figure 5: Local index (LI): estimated elapsed time for 50 Trec3 queries.

by increasing the number of machines in the network (while keeping the collection size unchanged), we obtain a speed up in the query processing which is limited, at the bottom, by the time to extract data from the disk (basically seek time).

In figure 6 we show, for the global index organization, how the estimated time to process the 50 test queries varies with the number of machines in the network. Again, we observe

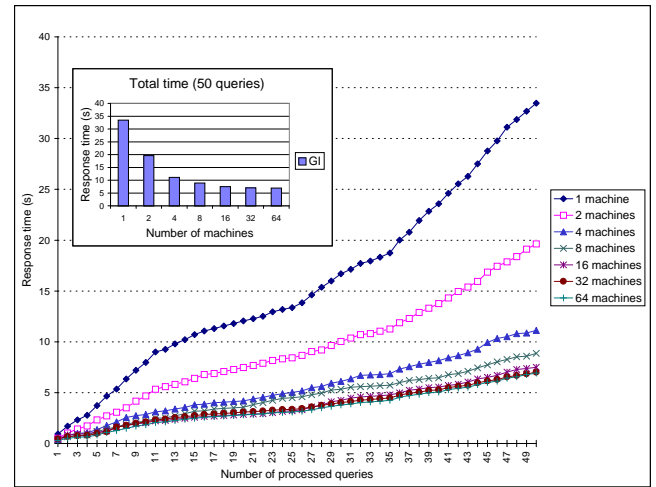


Figure 6: Global index (GI): estimated elapsed time for 50 Trec3 queries.

that by increasing the number of machines in the network (while keeping the collection size unchanged), we obtain a speed up in the query processing time which is limited, at the bottom, by the time to send the retrieved document numbers across the network and to sort them at the broker.

In figure 7, we compare the 50 queries total processing time for the global index and the local index organizations. As it

Number of machines	Response time (s)		GI as percentage of LI (%)
	LI	GI	
2	21.26	19.64	92.37
4	17.13	11.13	64.98
8	14.58	8.86	60.78
16	13.11	7.50	57.23
32	12.23	7.00	57.21
64	11.78	6.93	58.83

Figure 7: Global versus local index: estimated total time for 50 Trec3 queries.

can be seen, the global index organization consistently outperforms the local index organization at this network speed. Further, the relative improvement in performance increases with the number of machines in the network. The qualitative explanation is as follows.

With the global index organization, each local disk does not need to perform as many seek operations as it is necessary with the local index organization. The reason is that each machine has to process only the subquery relative to its portion of the global list. As the number of machines exceed the average query length (in number of keywords), each machine

tends to have to process only 1 or 2 terms locally and thus, only 1 or 2 seeks need to be performed in the local disk on the average.

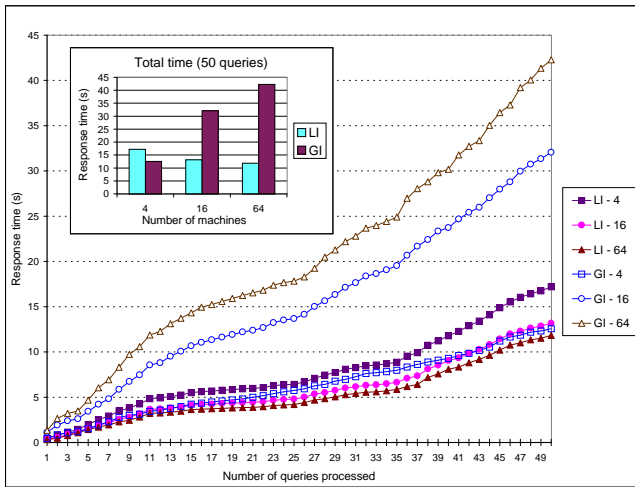


Figure 8: Slow network: estimated time for 50 Trec3 queries ( $tt = 1 \times 10^{-6}s$ ).

The fact that, for the global index organization, the number of seeks performed locally drops with the increase in the number of machines in the network is a crucial point. When the number of machines is large enough, each machine has to process only a single query keyword. However, since it has no other information on the documents in the answer set, it has to send the entire inverted list corresponding to that query keyword to the central broker. Thus, the positive point is that the number of seeks performed locally drops to a minimum. On the negative side, network traffic might increase considerably because larger lists of documents have to be sent to the central broker. We say that the global index organization allows trading disk accesses to network traffic (a fact also observed in [18]). Depending on the size of the queries, the size of the collection, and the speed of the network, such trading might become quite advantageous as illustrated in figure 7.

In figure 8, we compare the performances for the global index and the local index organizations considering a much slower network (10 times slower). The results shown are for a network speed of 8M bits per second which is typical of ethernet technology. As it can be seen, the global index organization no longer outperforms the local index organization consistently. Further, increasing the number of machines favors the local index organization because moving data between machines in this case is quite expensive.

In figure 9, we compare the performances for the global index and the local index organizations considering the presence of faster disks. The results shown are for a disk 4 times faster (and seek time as before i.e.,  $t_s = 10 \times 10^{-3}s$ ). As it can be seen, the global index organization is highly favored by faster disks because its disk access patterns are composed of fewer seeks and larger data transfers than the local index

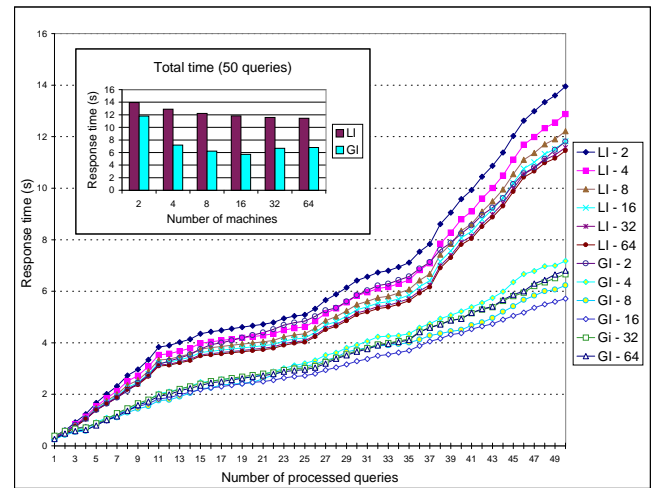


Figure 9: Fast disk: estimated elapsed time for 50 Trec3 queries ( $tr = \frac{4.64}{4} \times 10^{-7}s$ ).

organization.

## Conclusions

We studied the query performance for a digital library distributed in a tightly coupled environment. In our library, inverted files are used as index structures and the vector space model is adopted as ranking strategy. We consider both a local index organization and a global index organization.

Our study is based on an analytical model which is coupled with a small simulator. Our results indicate that, for Trec3 queries, a global index organization outperforms a local index organization consistently in the presence of fast communication channels. This is due mainly to two factors. First, the global index organization allows trading seek operations in disk to network traffic. Second, the global index organization allows greater concurrency among the various queries.

In the near future, we intend to evaluate the behavior of our distributed digital library in the presence of very short queries as those found in the Web. Furthermore, we intend to consider scalability issues regarding the size of the collection and the size of the queries.

## REFERENCES

1. I.J. Aalbersberg and F. Sijstermans. High-quality and high-performance full-text document retrieval: the parallel infoguide system. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 151–158, Miami Beach, Florida, 1991.
2. T. Anderson, D. Culler, and D. Patterson. A case for NOW (network of workstations). *IEEE Micro*, 15(1):54–64, February 1995.
3. F.J. Burkowski. Retrieval performance of a distributed text database utilizing a parallel processor document server. In *Proceedings of the Second International Sym-*

- posium on Databases in Parallel and Distributed Systems*, pages 71–79, Dublin, Ireland, 1990.
4. B. Cahoon and K.S. McKinley. Performance evaluation of a distributed architecture for information retrieval. In *Proc of the 19th ACM SIGIR Conference*, pages 110–118, Zurich, Switzerland, 1996.
  5. W. Frakes and R. Baeza-Yates, editors. *Information Retrieval – Data Structures & Algorithms*. Prentice Hall, 1992.
  6. D. Harman. Relevance feedback revisited. In *Proc of the Fifteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 1–10, Denmark, 1992.
  7. D. Harman. Overview of the third text retrieval conference. In *Proceedings of the Third Text Retrieval Conference - TREC-3*, Gaithersburg, Maryland, 1995. National Institute of Standards and Technology. NIST Special Publication 500-225.
  8. D. Harman, E. Fox, R. Baeza-Yates, and W. Lee. Inverted files. In W.B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures and Algorithms*, chapter 3. Prentice Hall, 1992.
  9. J. Heaps. *Information Retrieval - Computational and Theoretical Aspects*. Academic Press, NY, 1978.
  10. B.S. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):142–153, February 1995.
  11. Z. Lin. Cat: An execution model for concurrent full text search. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, Miami Beach, Florida, 1991.
  12. I.A. Macleod, T.P. Martin, B. Nordin, and J.R. Phillips. Strategies for building distributed information retrieval systems. *Information Processing & Management*, 23(6):511–528, 1987.
  13. National Institute of Standards and Technology. Proceedings of the text retrieval conference (trec), November 1992.
  14. M. Persin. Document filtering for fast ranking. In *Proc. of the 17th ACM SIGIR Conference*. Springer Verlag, 1994.
  15. G. Salton and C. Buckley. Term-weighting approaches in automatic retrieval. *Information Processing & Management*, 24(5):513–523, 1988.
  16. G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Co., New York, 1983.
  17. C. Stanfill, R. Thau, and D. Waltz. A parallel indexed algorithm for information retrieval. In *Proc. of the 12th ACM SIGIR Conference*, pages 88–97, 1989.
  18. A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Proceedings of the International Conference on Parallel and Distributed Information Systems*, 1993.