Improving Web Searches with Distributed Buckets Structures

V. Gil Costa, A. M. Printista
LIDIC - Computer Science Department
University of San Luis
San Luis, Argentina
{gvcosta,mprinti}@unsl.edu.ar*

M. Marín
Center of Web Research
University of Magallanes
Punta Arenas, Chile
mmarin@ona.fi.umag.cl

Abstract

This article compares several strategies for searching in Web engines and we present the bucket algorithms to improve the efficiency of a classical index data structure for parallel textual database. We use the inverted files as the data structure and the vector space model to perform the ranking of documents. The main interest is the queries parallel processing on a cluster of PCs, and therefore this paper is focused in the communication and synchronization optimization. The design of the server that processes the queries, is effected on top of the Bulk Synchronous-BSP model of parallel computing, to study how query performance is affected by the index organization.

1. Introduction

Few Web services require as much computation per request as search engines. These are of the major tools for information access. Most search engines use information retrieval techniques to rank web pages in presumed order of relevance based on a simple query. Compared to the bibliographic information retrieval systems of the 70s and 80s, new search engines must deal with information that is much more heterogeneous, messy, more varied in quality, and vastly more distributed or linked. In current Web environment, queries tend to be short (1-2 words) and the potential database is very large and growing rapidly. Estimates of the size of the Web range from 500 million to a billion pages, with many of these pages being portals to other databases (the hidden Web).

Due to the enormous amount of information, search engines have become indispensable for finding specific and appropriated information and in response to this huge expansion of potential information sources, today's Web search engines have emphasized speed and coverage, with less importance attached to effectiveness. Because of this, several studies have been met to the development of new strategies that allow satisfying these demands through the parallel processing; this parallel processing has demonstrated to be a paradigm that allows to improve the algorithms execution times.

For efficient querying, specialized indexing techniques have to be used with large document collections. 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 [6]. Each of them has their own strong and weak points. However, due to its simplicity and good performance, *inverted files* or *inverted lists* have been traditionally the most popular indexing technique used along the years. Therefore, in this work, we consider that the document collection is indexed using inverted files.

Assuming a text collection composed of a large set of documents, an inverted list is basically composed of a table (the vocabulary) that maintains all the relevant words found in the text, and an associated list for every such word that registers all occurrences of the word in the text (document-id and another information used to rank out responses to users queries) [5].

Because the user does not exactly understand the meaning of searching using a set of words, and he may get unexpected answer, because he is not aware of the logical view of the text adopted by the system and finally because he has trouble with the boolean logic, is why these algorithms use single key models (vector space model [6]).

The rest of the paper is organized as follows. In section 2 a review of previous work is presented. Section 3 presents the parallel model BSP. Section 4 describes the server's architecture. In section 5 we describe the structure and the model used to rank out the queries. In section 6 we present the iterative bucket inverted files strategy and the analytical model is shown in the next section. The results are showed in section 8. Conclusions and future research directions are in section 9.

2. Previous work and motivation

The work presented in [26] compares the impact of performance for queries processing, using two different organizations for the invested lists. It proposes two basic options to classify the indexes: disk index and system index. In the disk organization (or local organization), documents are evenly divided in groups, one for each disk, and in each partition the inverted indexes are built from the documents that reside there. In the system index organization, well known as global organization, the complete index is evenly distributed through the disks. Queries were boolean and the architecture is a LAN, where the number of CPUs, the number of in/out CPU controllers, and the number of disks varies.

The work performed in [19] uses the probabilistic model, a model that does not takes into account the frequency of occurrence of each term of the index in the document [6]. It is also considered that users usually send small quantities of queries, and therefore only one or two processors of the system will be lending service, and the remaining ones will be lazy.

The works presented in [3, 4] use the vector space model to perform the ranking operation, and the experiments are performed with the collection TREC-3 [14] and their group of queries. It also implements and evaluates the performance of the query execution on real and contrasted cases. Here the partitioned inverted lists using the local or global organization are proposed to reduce the data communication between the processors. But they have to estimate some values to perform a distributed filter technique in the local case, and in the global case the amount of data been communicated depends on the number of processors. Besides, only one query is solved at the time. The work presented in [18] also performs an analysis over this structure.

The works in [9, 10, 11] present a parallel focus for the local and global organization, using the BSP model. A performance study of these indexation strategies is performed and an analytic analysis based on the BSP cost model is presented.

Now, these two classic index organizations have some problems. With the local strategy the problem arrives when we use small databases because the broadcast performed by this strategy is too expensive. And the problem with the global strategy is that inverted files have associated lists of different large, so at the queries processing time some machines will have more work to do than others will. A first approach to solve these problems was proposed in [20] through the Composite Inverted Files. But this work is very extremist because every term is treated as local or global. Therefore, in this work we try to find a middle situation between the local and global organization using the BSP model to analyze their costs.

3. Computation model

In the *Bulk Synchronous Parallel, BSP* model of computing, proposed in 1990 by Leslie Valiant [27], any parallel computer is seen as composed of a set of P processor-local-memory components which communicate with each other through messages. The computation is organized as a sequence of $supersteps$. During a superstep, the processors may perform sequential computations on local data and/or send message to others processors. The messages are available for processing at their destination by the next superstep, and each superstep is ended with the barrier synchronization of processors [24].

The practical model of programming is SPMD, which is realized as C and C++ program copies running on P processors, wherein communication and synchronization among copies are performed by ways of libraries such as $BSPlib$ [15] or $BSPpub$ [17].

BSP is actually a parallel programming paradigm and not a particular communication library. In practice, it is certainly possible to implement BSP programs using the traditional PVM [8] and MPI [25] libraries.

The BSP model establishes a new style of parallel programming to write programs of general purpose, whose main characteristic are its easiness and writing simplicity and its independence of the underlying architecture (portability). BSP achieves the previous properties elevating the level of abstraction which programs are written with [12, 13].

The total running time cost of a BSP program is the accumulative sum of the cost of its supersteps, and the cost of each superstep is the sum of three quantities: w, hG y L, where w is the number of operations performed. h is the maximum of messages sent/received by each processor with each word costing G units of running time, and L is the cost of barrier synchronizing the processors. The effect of the computer architecture is included by the parameters G and L, which are increasing functions of P. This values along with the processor's speed s (e.g. mflops) can be empirically determinate for each parallel computer by executing benchmark programs at installation time.

4. Server's architecture

The environment selected to process the queries is a network of workstations connected by fast switching technology. A network of workstations is an attractive alternative nowadays due the emergent fast switching technology provides fast message exchanges and consequently less parallelism overhead [1, 28, 29].

To process the queries coming from different users, the server has to access the textual database. This server has P

processors and at least one $broker$ machine that acts as middleman between the server's processors and the users. The queries coming from the users are received by the $broker$ machine which should route them, with some methodology, to the server.

Also, for each query received, one of those P server's processors will be the $ranker$ machine, which is selected by the $broker$ during the queries distribution time. This $ranker$ machine, will gather the partial results, will perform the final ranking of document identifiers and will send a reply to the requesting user machine.

5. Distributed Inverted Files and Vector Space Model

In this section we discuss about the choice of the inverted files as the index structure, how this index is built and distributed across the server. About the vector space model ranking strategy and the queries processing.

Inverted files are useful because their searching strategy is based mostly in the vocabulary, which usually fits in main memory. Further, inverted files are simple to update and perform well when the pattern to be search for is formed by conjunctions and disjunction of simple words, which is probably the most common type of query in information retrieval systems.

An inverted file has a list of all distinct words that appears in the documents of the database, that is known as the vocabulary, and each word or term in the vocabulary has an associated list of documents in which the word occurs. The vocabulary is sorted in lexicographical order. Since we use the vector space model to rank documents in the query answer set, the associated lists keep information about the frequency of the word in the documents. Therefore, the associated lists have pairs of two things, one of them is the document identifier where the term appears, and the other one is the frequency of the word in that document $< d_{id}, f_{t,d} >$).

With large text some restrictions are imposed on the inverted file to keep it smaller [7]. Some of these restrictions are filtering of text characters and separators, use of controlled vocabulary in which not all words in the text are indexed, and exclusion from the vocabulary of stop words such as articles and prepositions.

In this work we adopt the vector space model as the ranking strategy. This is the most popular model of information retrieval proposed by Salton G. at the beginning of the 70 years [23] and broadly diffused from then on. This model's base consists on the document representation through vectors. In those vectors each element would be an evident characteristic in the documents. An elementary vision of the concept "characteristic" is the equivalent to word.

In general words, the idea is that the weight of a term in a given document is inversely proportional to its frequency in the document collection and directly proportional to its frequency in the document in question. There are several formulas proposed to calculate the weight, based on these ideas.

The vector space model proposes a mark, in which the partial coincidences are possible assigning non-binary weights to queries terms and to the documents. These weights are used to compute the grade of similarity among each document stored in the system and the user's queries. Ordering the recovered documents in falling order by this grade of similarity, the vector space model considers the documents that coincide partially with the terms of the query. The main resulting effect is that the group of answer of ranked documents is more precise.

To perform a ranking of documents we have to [6]:

- Select the terms from the documents.
- Each word is a term except the $stopwords$.
- Let be $\{t_1...t_n\}$ the group of terms and $\{d_1...d_n\}$ the group of documents, a document d_i is seen as a vector:

  $$d_i \rightarrow \vec{d_i} = (w(t_1, d_i), ..., w(t_k, d_i))$$

  where $w(t_r, d_i)$ is the term weight t_r in the document d_i.
- In particular, documents can be seen as queries and therefore as vectors.
- The similarity between a query q and a document d is:

  $$0 <= sim(d, q) = \sum_t (w_{q,t} * w_{d,t})/Wd <= 1.$$

  The similarity is calculated among a query q and a document d as the difference cosine that geometrically corresponds to the cosine of the angle among the two vectors. The similarity is a value between 0 and 1. Notice that the same documents have similarity 1 and if they do not share terms they have similarity 0. Therefore this formula allows to compute the relevance of the document d for the query q.
- This formula reflects the weight of the term t in the document d (that is to say how so important it is this term for the document).

  $$0 <= w_{d,t} = f_{d,t}/max_k * idf_t <= 1.$$
- $f_{d,t}/max_k$ it is the normalized frequency, $f_{d,t}$ is the number of repetitions of t in d. If a term appears many times in a document, it is supposed that it is important for that document, therefore $f_{d,t}$ grows. max_k it is the frequency of the most popular term in the document d. This formula is divided by max_k to normalize the vector and to avoid favoring longest documents.

- $idf_t = log10(N/nt)$, where N is the number of documents in the collection, nt is the number of documents where t appears.

- $Wd = (\sum(w_{d,t}^2))^{1/2}$. Is used as the normalization factor. It is the weight of the document d in the document collection.

- $w_{q,t} = (f_{q,t}/max_k) * idf_t$, where $f_{q,t}$ is the term frequency t in the query q, and max_k is the frequency of the most popular term in the query.

Ranking techniques are effective to find answers in document collections, but they can be expensive of evaluating. The use of filters [22] is an evaluation technique that allows to recognize what documents are probable of being ranked, reducing the main memory volume required and the queries evaluation time, without harming the effectiveness of results. Because of that, we use the filtering technique proposed by Persin [21] to discard the irrelevant documents for the queries. It uses two filters: the f_{ins} to reduce the amount of memory used and f_{add} to reduce the CPU time. This filter technique avoids traveling the entire associated list of a term appearing in the query.

In the next section we introduce the iterative bucket inverted files in order to find a better solution to Web search problems, looking for a middle situation between the local and global index strategies.

6. Iterative Bucket Inverted Files

In order to reduce the amount of communication between the processors and to avoid the problems of the local and global strategies, we can use the iterative bucket inverted files. The main idea of the buckets is to obtain a better distribution of the index finding a middle situation between the local and global organization, and in this way we can improve the load balancing during the queries processing time.

There are many ways to distribute the terms of the vocabulary table with its documents identifiers and frequencies. Here we present two alternatives: one of them is a hash distribution and the other one is a multiplexed distribution. In the first case, we divide the associated list of each term in buckets, where the first ones will have documents with higher frequencies than the last ones. Then we send these buckets to the processors using the hash function with three parameters: the term, the bucket identifier and the number of processors. In this distribution not all processors will get a part of the associated list because it depends on the bucket size.

In the multiplexed distribution, we also divide the associated lists in buckets of fixed sizes $B = N/P$, where N is the number of pairs in the associated lists, and P is the number of processors in the server. Here, each pair of the bucket is distributed in a circular way through the processors, as shown in Figure 1, so every processor will have a portion of the associated list of each term.

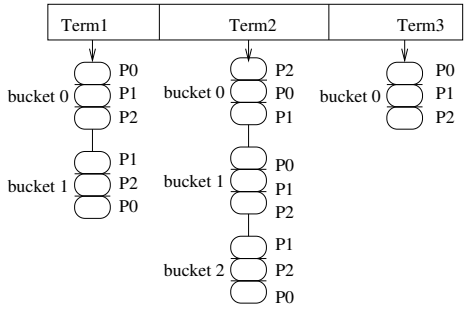

Figure 1. Multiplexed distribution. A server with three processors, each one has a portion of a bucket

To process queries using the iterative bucket hash distribution we require only two supersteps. In the first one, the broker machine sends a sub-query with the corresponding terms to the server processors that have information about them. These processors get only a part of the associated list of each term appearing in the query (K, where K is the number of answers to the user query). So processors get K pairs of document-identifier and frequencies, and send them to the ranker machine. They also send the next pair of document-identifier and frequency.

This last machine will perform, in the second superstep, the ranking of documents checking out if the document after the K pairs sent is relevant to the query. If so, it will have to ask to the remittent processor for others K pairs. We can know if the documents that remains in the processors are relevant through the filtering technique applied in the ranking algorithm.

On the other hand, the multiplexed distribution requires three supersteps. Remember that all processors have a portion of the associated list of each term. Then, to perform the queries processing operation the broker machine sends the whole query to one machine in the first superstep. This last one will broadcast the query. In the second superstep, when all processors have the query, they will search in their inverted lists the K most important documents and will send them to the ranker machine with the next pair in the associated list. In the third superstep the ranker machine will perform the ranking of documents, checking out if some processor has relevant documents that have not been send. If that happens, the ranker will have to send a message to those processors and will have to wait for the next K pairs.

The iterative strategy hypothesis is the communication

and compute cost reduction when the textual database is bigger enough.

7. Analytical Model

In this section we present an analytical analysis using the BSP computing model, for our iterative bucket strategy. The processing cost is considered since the queries arrive to the server until the ranker processor sends out the responses to those queries. Secondary memory is treated as a communication network, so there is a parameter D to represent the average cost of accessing to secondary memory. This parameter can be easily obtained using *benchmark* programs from the Unix systems. If the whole database can be stored in main memory, then $D = 1$.

The execution of a batch $Q = qP$ of queries using the multiplexed distribution is as follows. In the first superstep, the processors get q queries and broadcast them with a cost of qPG, after that a barrier synchronization is performed. So this superstep cost is $t_1 = qG + qPG + L$, where L is the synchronization cost, qP is the maximum number of messages send/received, and G is the cost in words of sending the message.

In the second superstep, the processors get $qP = Q$ queries, and recover from the inverted list the K most relevant pairs document-frequency with a cost of $q(K)D$. Due to these algorithms are iterative we have to add the cost of asking if some processor has more relevant documents, and this cost is $P(W/P)G$.

Finally, in the third superstep, the ranker processors gets P message of size K, and performs the ranking with a cost of PK. So, this superstep has a cost of $t_3 = KPG + KP + KG + L$.

The iterative bucket hash distribution has a similar analysis but we have to omit the analysis of the first superstep because this distribution does not perform a broadcast. In other words, some processors get a sub-query with terms of the original query that they have in their inverted files. Then they select the top K pairs of the associated lists and send them to the ranker machine. In the second superstep, the ranker machine receives message of size K and perform the ranking operation. The asymptotic analysis is shown in table 1.

Another important aspect to be analyzed is the data distribution. If the vocabulary table size is T and the number of pairs in each list of a term is approximately N, the multiplexed distribution requires $T * \rceil N/P$ in each processor. Where $K = \rceil N/P$ is the bucket size. In this case, each processor receives only one bucket for each term. While in the hash distribution the space required in each processor varies due to the bucket size. We have three possible cases. In the first one, if the number of processors P is equal to the number of buckets $K = \rceil N/P$, then each machine will require a space of size $T * K$. In the second case, if $B < P$ only some processors will get buckets of a term and the space required is $T/P * K$. Finally in the last case, if $B > P$ all processors will get a bucket of the terms, and some of them may get more than one ($\rceil B/P$). So, the space required here is $T * [K * \rceil B/P]$.

Table 1. Asymptotic cost for the iterative buckets distributions

Multiplexed	$K(qD + P) + (qP + K(q + P) + W)G + L$
Hash	$q(cK)D + cK + (q(cK) + W + K)G + L$

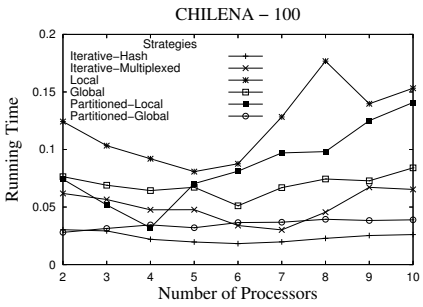

Figure 2. Running time in microseconds for the presented strategies using the Chilean textual database for a batch of 100 queries

8. Results

In this section we investigate the system performance using a 2GB sample of the Chilean Web with a query log from www.todocl.cl [16]. This gave as a realistic setting both on the set of term that compose the text collection and the type of term that typically are part of user's queries. Transactions were generated at random by taking terms from the query log. The presented strategies are compared with the classical local and global index organization and we also implement under the BSP model the partitioned inverted lists that use a distributed ranking algorithm [3]. The experiments were performed with a filter Cins=0.12 and Cadd=0.335 [2]. The developments were performed in a cluster of 8 SMP (dual), connected by a 1000Mbs (1000BASE-T) Ethernet and the time was measured in microseconds.

Figure 2 and 3 show the running time for the classical local and global index organization, for the partitioned inverted files implemented under the BSP model, and for our

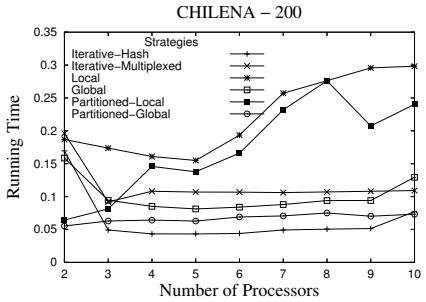

Figure 3. Running time in microseconds for the presented strategies using the Chilean textual database for a batch of 200 queries

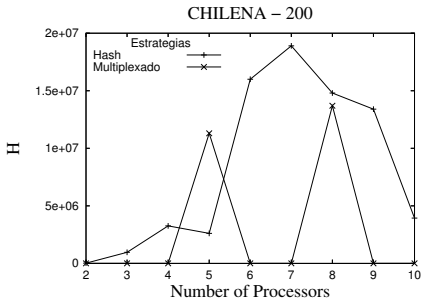

Figure 4. Communication cost obtained for the execution of a batch of 200 queries

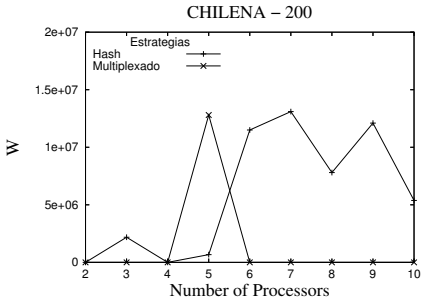

Figure 5. Computation cost for the presented strategies.

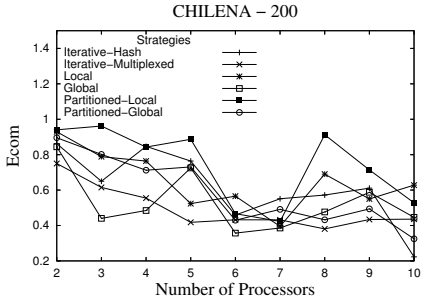

Figure 6. Communication efficiency obtained for the execution of a batch of 200 queries

proposed work: the iterative bucket strategies. As you can see, the iterative hash algorithm performs better than the other ones. That is because it has a better data distribution among the processors finding a middle situation between the classical organizations. On the other hand, due it is iterative we can reduce the data communication among processors.

Figure 6 shows the communication cost:

$$W = \sum_{SS} messages/Total\_SS \quad (1)$$

and Figure 7 shows the computation cost:

$$H = \sum_{SS} operations/Total\_SS \quad (2)$$

where the operations of this last formula are: recovering the pairs documents-identifier and frequencies, ranking, sort operations, etc. This last one allows measuring the loadwork in the processors. With the multiplexed distribution the W cost is lower because the data is more distributed. But with the hash distribution not all processors have information about the queries, so we have a poor load balancing. Analyzing the H cost we have a similar behavior, due in the hash case the information is more concentrated in a few processors needing to perform more iterations to get a right answer.

Figure 6 shows the communication efficiency (see equation 3), and Figure 7 shows the computation efficiency obtained by the equation 4, where the operations are: recovering the pairs documents-identifier and frequencies, the ranking, sort operations, etc. This last one allows measuring the loadwork in the processors. As the number of the server machines grows up, this efficiency tends to go down getting closer to zero, due to some processors will get less work to do than others. $Ecomp = 1$ indicates a good balance. In these figures we can see that the strategies using a local organization obtain better computation efficiency, than the others do. While the global organization strategies get lower values of $Ecom$.

$$Ecom = \frac{\frac{\sum_P messages\ send/received}{P}}{Maximum\ amount\ of\ messages\ send/received} \quad (3)$$

$$Ecomp = \frac{\frac{\sum_P operations}{P}}{Maximum\ amount\ of\ operations} \quad (4)$$

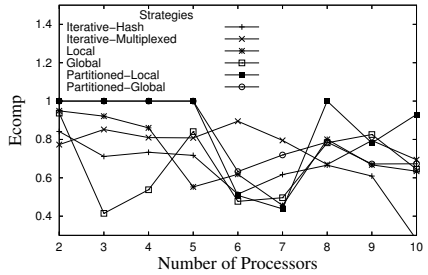

Figure 7. Computation efficiency for the presented strategies.

9. Conclusions and Future Work

We studied the query performance for a textual database distributed in a cluster of PCs. In the algorithms implemented, the vector space model is adopted as the ranking strategy, and the inverted files are used as index structures. In our experiments we consider the classical local and global organizations, and the partitioned strategies presented in previous works. We compared them with our iterative bucket strategies.

Our study is based on the BSP computing model, to predict the costs of these algorithms. The experimental results obtained indicate that the iterative bucket hash distribution outperform the classical index organizations and gets a better running time than the partitioned strategies. This is mainly because this distribution does not require sending the whole pairs for each term in the query. Another important advantage is that we do not have to estimate any value to execute the filtering technique in the ranking operation.

As future we are going to study how to perform multimedia searches in a Web engine, using specific data structure based on pivots and partition such as the sat, the dynamic version of this structure, the GNAT, etc.

References

[1] T. Anderson, D. Culler, D. Patterson, and the Now Team. A case for now (network of workstations). Technical report, IEEE Micro, 15(1), 1995.

[2] C. Santos Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Concurrent query processing using distributed inverted files. *In the 8th. International Symposium on String Processing and Information Retrieval*, pages 10–20, 2001.

[3] Claudine Santos Badue. Processamento distribuído de consultas usando arquivos invertidos particionados. Master's thesis, Universidade Federal de Minas Gerais, Istituto de Ciencias Exatas, Departamento de Ciencia de Computaçao, 2001.

[4] Claudine Santos Badue, Ricardo A. Baeza-Yates, Berthier A. Ribeiro-Neto, and Nivio Ziviani. Distributed query processing using partitioned inverted files. In *SPIRE*, pages 10–20, 2001.

[5] R. Baeza-Yates, A. Moffat, and G. Navarro. *Searching Large Text Collections*, pages 195–244. Kluwer Academic Publishers, 2002.

[6] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ISBN:0-201-39829-X, 1999.

[7] R. Baeza-Yates D. Harman, E. Fox and W. Lee. *Inverted Files*. In W.B. Frankes and R. Baeza-Yates editors, Information Retrieval: Data Structures and Algorithms, chapter 3. Prentice Hall, 1992.

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users Guide and Tutorial for Network Parallel Computing*, 1994. MIT Press.

[9] V. Gil-Costa. *Procesamiento Paralelo de Queries sobre Base de Datos Textuales*. Tesis de Licenciatura. Universidad Nacional de San Luis, Argentina, 2003.

[10] V. Gil-Costa, M. Printista, and M. Marín. Parallel algorithms in the web. *WWW/Internet 2005 in Lisboa, Portugal.ISBN: 0-7695-2471-0*, 19-22 October, 2005.

[11] V. Gil-Costa, M. Printista, and M. Marín. Un enfoque paralelo para el procesamiento de consultas sobre base de datos textuales. *IV Workshop de Procesamiento Distribuido y Paralelo (WPDP). IX Congreso Argentino de Ciencias de la Computación (CACIC 2003)*, pages 445–458, Octubre 2003.

[12] M. Goudreau, J. Hill, K. Lang, B. Mc Coll, and S. Rao. *A Proposal for the BSP Worldwide Standar Library*. http://www.bsp-worldwide.org/standar/stand2.html, 1996.

[13] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas. Towards efficiency and portability: Programming with the bsp model. In *SPAA'96: 8th Annual ACM SPAA*, pages 1–12, 1996.

[14] Donna Hawking. Overview of the third text retrieval conference. *Proceedings of the third text Retrieval Conference (TREC-3)*, pages 1–19, Maryland, U.S.A.,1994. NIST Special Publication 500-207.

[15] http://www.bsp worldwide.org. Bsp worldwilde standard.

[16] http://www.todocl.cl. Todocl search engine main page.

[17] http://www.uni paderborn.de/bsp. Bsp pub library at paderborn university.

[18] A. MacFarlane, J.A. McCann, and S. E. Robertson. Parallel search using partitionated inverted files. *In Proceedings of the 7th International Symposium on String Processing and Information Retrieval*, pages 209–220, La Coruna, Spain. September 2000. IEEE Computer Society.

[19] A. MacParlane, J.A.McCann, and S.E. Robertson. Parallel search using inverted files. *7th. International Symposium on String Processing and Information Retrieval*, 2000.

[20] M. Marín. Parallel text query processing using composite inverted lists. In *Proceedings of the Second International Conference on Hybrid Intelligent Systems (Invited session on Web Computing)*, Dec. 2002.

[21] M. Persin. Document filtering for fast ranking. *In Proceedings of the 17th ACM SIGIR Conference*, pages 339–348, Dublin, Irland. July, 1994.

[22] M. Persin, J. Zobel, and R. Sacks-Davis. Filteres document retriaval with frequency-stores indexes. *Journal of the American Society for Information Science*, 1996.

[23] G. Salton and M. McGill. *Introduction to Modern Information Retrieval,New York*. McGraw-Hill, 1983.

[24] D.B. Skillcorn, J. Hill, and W.F. McColl. Questions and answers about bsp. Technical Report PRG-TR-15-96, Oxford University Computing Laboratory, 1996.

[25] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI: The complete Reference*, 1996.

[26] Anthony Tomasic and Hector García-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. *In Proceedings of the Second International Conference on Parallel and Distributed Information Systems. San Diego, California, U.S.A.*, pages 8–17, 1993.

[27] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103:111, 1990.

[28] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Aplications Using Networked Workstations*. Prentice Hall, 1997.

[29] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Aplications Using Networked Workstations*. Prentice Hall, 1999.