CORE
Metadata, citation and similar papers at core.ac.uk
Provided by El Servicio de Difusión de la Creación Intelectual

# A Parallel View for Search Engines

**Gil Costa V., Pérsico A., Printista M.**
Computer Science Department
University of San Luis
Ejército de los Andes 950
5700 - San Luis, Argentina
{gvcosta,persicoa,mprinti}@unsl.edu.ar

**Mauricio M.**
Computing Department
University of Magallanes
Punta Arenas, Chile
mmarin@ona.fi.umag.cl
*

VI Workshop de Procesamiento Distribuido y Paralelo (WPDP)

**Abstract**

To engineer a search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of n-tillions of queries every day. Despite the importance of large-scale search engines on the Web, very little academic research has been done on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from years ago. In most papers the index simply "is", without discussion of how it was created. But for a indexing scheme to be useful it must be possible for the index to be constructed in a reasonable amount of time, and so papers describing complex indexing methods should also describe and analyze a mechanism whereby the index can be built. Scalability is of concern during index construction as well as during query processing. This paper describes the cooperative work between the Crawler, Indexer and the Searcher.

**Key-Words: Search Engine, Web, Crawler, Index, Searcher, Inverted Lists, BSP**

## 1 Introduction

The importance of text retrieval systems in Internet has grown dramatically during recent years due to very rapid increase of available storage capacity, increased performance of all types of processors and exponential growth of the global networks that provide an enormous source of different documents. One of the critical factors for usability of text retrieval systems is the performance of search engine and underlying indexing techniques [12].

*Group supported by the UNSL and the ANPCYT (National Agency for Promotion of the Science and the Technology)

The search engines are systems that index automatically a portion of pages from the whole Web and allow to locate information through the formulation of a question. The search engines also manipulate great databases of references to web pages that have been created through an automatic process, without human intervention and generally of greater size. The search engines don't have subcategories like Directory Services, but they have advanced algorithms search that analyze the pages that they have in memory and with it they provide the most suitable result to a search (for example Google).

The main topics to consider at the time of evaluating the quality of a search engine are [2]: the number of documents of Internet stored in the index, the flexibility and quality of the query language, the correct results (noise and silence), the added services that incorporate in the search engine, the index update frequency, and the recovery speed and the connection difficulties.

The search engines are more exhaustive than the indexes about the volume of stored pages (several dozens of millions opposed to few hundreds of thousands), but they are much less necessary than the indexes, because their content is not a human indexing objet. A conventional search engine is composed by three main elements [14]:

- Crawler: process which recovers information from the Web.
- Indexer: process which organizes the information of the Web.
- Searcher: process which solves thre user's queries.

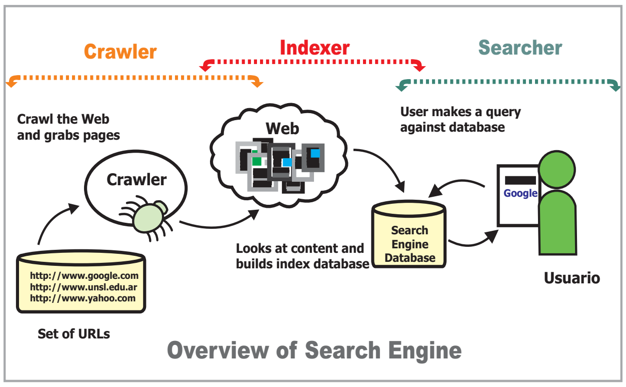

Figure 1: Overview of a Search Engine.

The relation between the main elements that compose a conventional crawler are presented in Figure 1. According to this model the efficiency and scalability of a search engine are related to the crawlers, which maintain updated the index on which the engine works. During the past years, different types of index to retrieval text were proposed, investigated and extensively used: techniques based on inverted lists.

The inverted lists provide reasonably good performance in single-keys searches (logarithmic on the database size, which actually means few disk accesses per search), but their performance rapidly degrade when the query size increases, which is of major importance for text retrieval. Other researchers have tried to process this data structure in parallel, using traditional models of parallel computing such as message passing computing through $PVM$ or $MPI$ [17] [18]. These experiments have proven that this structure can be processed in parallel in an efficient way.

In this paper a parallel implementation of this structure is presented using the *Bulk Synchronous Parallel, BSP* model of computing, proposed in 1990 by Leslie Valiant [16]. Also the relationship between the different existing modules in a search engine, crawler-indexer-searcher, is described emphasizing the cooperation among them, analyzing the effects on the performance of the engine according to the different decisions that are taken on each module.

## 2 Model of Computing

In the $BSP$ model any parallel computer is seen as composed of a set of $P$ processor-local-memory components which communicate with each other through messages. A $BSP$ computer is characterized by the bandwidth of the network, the number of processors, its speed and for the synchronization time among all the processors. All these characteristics are part of the parameters of a $BSP$ computer.

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, its independence of the underlying architecture (portability). $BSP$ achieves the previous properties elevating the level of abstraction with which the programs are written.

The computation is organised 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 message are available for processing at their destination by the next superstep, and each superstep is ended with the barrier synchronization of processors [19, 20].

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 is performed by ways of libraries such as *BSPlib* [14] or *BSPpub* [13].

## 3 Search Engine Architecture Overview

First, a high level discussion of the architecture will be explained and then the major applications: crawling, indexing, and searching will be deeply examined.

The main components of a search engine are:

- Crawler: traverses the Web looking for pages to be indexed.
- Indexer: manteins updated the index with that information.
- Searcher: performs the searches in the index.

This paper, is focused in centralized search engine architecture (ej. AltaVista), where the crawler (robot, spider) runs locally in the search engine, crossing the Web by means of orders to the web servers and bringing the text of the Web pages that goes finding.

The indexer is used in centralized form to respond the queries submitted from different places in the Web, however it can work in a parallel way. Also, the searcher runs in the local system and makes the searches in the index, returning the ranking URLs.

To process the queries coming from different users, the searcher has to access the textual database, and for it the searcher is formed with $P$ 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 to a *victim* machine of the server with some methodology, depending on the distribution strategy in use.

Also, for each query received, one of those $P$ searcher's processors will be the *ranker* machine, which is selected by the *broker* during the queries distribution. This *ranker* machine will perform the final ranking of document identifiers, and will send it to the requesting user machine.

# 4 Web Crawler

Running a Web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application in the search engine since it involves interacting with hundreds of thousands of web servers and various name servers which are all beyond the control of the system.

The Web crawler retrieves a web page and all the web pages with which it maintains links, thus then the indexer, can index the information according to a predefined criterion. Some of the criteria that can be used are a document title, the put-data, the number of times that a word is repeated in a document, algorithms to value the relevance of a document, and the weight of each change criterion according to the profile of the search engine.

The basic crawling algorithm is: **(a)** fetch the page, **(b)** parse it to extract all linked URLs, and **(c)** for all the URLs not seen before, repeat steps [**(a)**-**(c)**].

In order to scale hundreds of millions of web pages, there are different crawling techniques. The simplest is to start with a set of URLs and from there extract other URLs which are followed recursively in a breadth-first or depth-first way. A variation is to start with a set of popular URLs because we can expect that they have information frequetly requested. Another technique is to partition the Web using country codes, domain names, etc.

There are different architectures to crawl the Web: centralized and distributed architectures. The first one consists of a crawler to traverse the Web in the system, and in the second, several crawlers coordinate to crawl the whole Web, each crawler run on distributed sites and interacts in a peer-to-peer fashion and each crawler has its own set of URLs. For this last, is difficult to avoid visiting the same page more than once.

However, crawling the Web is not a trivial programming problem but a serious algorithm and it has system design challenge because of the the following two factors. the first one is that *the Web is very large. Currently Google [9] claims to have indexed over 3 billion pages. Various studies [2] have indicated that, historically, the Web has doubled every 9-12 months.*, an the second one is that *Web pages are changing rapidly. If change is any change, then about* 40% *of the Web pages change weekly [4].*

The main functions relationed with a crawler are the statistical analysis that measures the growth of the Web, the number of connected servers, etc. And the maintenance of the hypertextual structure of the Web that verifies the correct links between documents and eliminating or keeping information from the denominated dead links, that is to say, web pages that already have disappeared.

## 4.1 Crawling Characteristics

It is important to know the way crawlers acts over the pages, since the success of the registry of the web site depends on it and to reach a good classification.

- Deep Tracking: the search engine lists many pages of a site, still if they are not explicitly registered in it.
- Frames Support: a characteristic that allows the search engine to follow the connections through the frames.
- Maps of images: they are connections to other pages through images.
- Robots.txt: a text file that allows to indicate that pages do not have to be indexed in the site.

- Meta index robot: it has the same objective that robots.txt, but this it is an instruction of HTML code of the page.
- Tracking by popularity connections: the popularity of a page is detected analyzing whichever connections exist towards another page. The search engine uses that characteristic to determine what pages must be included in the index of their database, although this not necessarily indicates that they will obtain a good classification.
- Learning by frequency: the search engine learns with which frequency the pages are modified, to estimate the time in which the crawler will visit them again.
- Paid Inclusion: it's shown if the search engine offers a program where it is possible to be paid to guarantee that the pages of a site are included in the index. This is not just like paid positioning, which in addition to the inclusion in the index, guarantees an individual position in relation to a searched term.

## 5 Indexer

The indexer uses the collected data from the Web to build the database. The query enables users to query the indexed database with the help of a query form.

There are two main indexing methods, inverted lists and signature files, that have been proposed for large text databases. Inverted file index and signature file index have similar requirements during index construction.

In the following sections the steps to build the Web indexing are explained.

### 5.1 Indexing the Web

*Parsing.* Any parser which is designed to run on the entire Web must handle a huge array of possible errors. These range from types in HTML tags to kilobytes of zeros in the middle of a tag, non-ASCII characters, HTML tags nested hundreds deep, and a great variety of other errors that challenge anyone's imagination to come up with equally creative ones.

*Indexing documents into inverted lists.* After each document is parsed, every word is converted into a wordID. New additions to the vocabulary table are logged to a file. Once the words are converted into wordID's, their occurrences in the current document are translated into hit lists. The main difficulty with parallelization of the indexing phase is that the vocabulary table needs to be shared. Instead of sharing the table, the table is splited and distributed among the processor. Additional computationg is made to reduce the operations during de ranking process. The inverted list is ordered lexicographically.

### 5.2 Indexing Characteristics

The indexing characteristics indicate what it is indexed when the search engine tracks the page.

- **Full-Text**: index all the visible text in the body of the page, although some do not index some words (stop words) or they exclude them to seem spam.
- **Stop Words**: some search engine exclude words when they index the page or they at least do not consider them during the query. These words are excluded to save space or to increase the speed search, since they are words that they pretend to be spam.

- **Meta description and puts key words**: are meta index that describe the content of the page and the terms with which it is associated to him for the search.
- **Alternative Text and Commentaries**: the alternative text is associated with an image to describe it briefly, the alternative text is part of language HTML. The commentaries usually are an annotation on the page and are a meta-index type.

The work presented in this paper index the whole text removing the stop words from it.

## 6 Searcher

The searchers are one of the most used tools in Internet, they are portals or virtual sites where the users look for a particular information on a topic. The information that the user receives from the searcher comes in form of connections or links to the documents that have more coincidence to his queries.

So the searcher's job is to solve the queries coming from the users connected to the Internet. To do that, they have to index the queries and then to select the best documents according to this queries. The most popular data structure used in this kind of server, are the inverted list which is a word-oriented mechanism for indexing a text collection in order to speed up the searching task. The inverted list structure is composed of two elements: the vocabulary table and the occurrences. The vocabulary table is a set of all different words in the text. For each such word a list of all the text positions where the word appears and the frecuency (numbers of times the word appears in the document) is stored. The set of all those lists is called the 'occurrences' or 'associated lists'.

The indexing operation can be made through different strategies: global, local, buckets distributed among different processors with random, hash, circulate or sequential distribution, and buckets distributed among different supersteps (according to the BSP model) [5, 6, 7, 8, 10, 11, 13]. This strategies allow to index the users queries and then to solve them in parallel in the server. In the local index strategy, each processor builds its own index using his local documents, while in the others strategies a vocabulary table with all the words of the whole documents collection is built, and then the words with these occurrences are distributed among the processors.

The queries processing consists on selecting the best documents for the queries. To do that, the server has to perform the ranking operation using the vectorial, boolean, probabilistic model or some alternative [1]. In the presented strategies, the vectorial model is used.

During the queries processing, the queries may have many words and only the $K$ most relevants documents are recovered. On the other hand, the documents for each term are stored in falling order of frecuency to speed up the ranking operation. Depending on the used strategy, it is possible to recover all the documents for the query, or just the most important in each processor (bucket strategies).

## 7 Cooperative Work

Measuring the Internet and in particular the Web is a difficult task due to its hightly dinamic nature. Now days, there are more than 40 millions computers in more than 200 countries connected to the Internet, many of them hosting web server. The estimated number of web servers ranges from 2.4 millions to over three millions. This wide range migth be explained when we consider that there are many web sites that shares the same web servers using virtual hosts, that not all of them are fully accesible, that many of them are provisional, etc.

Because the user does not exactly understand the meaning of searching using a set of words, and the user 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 web searchers use single key models (vectorial model).

Recent studies have shown statisticals about the fashion in which the Web users make a query in a searhc engine:

- 80% of the queries do not use any type of operator.
- 25% of the users normally use a single word in their query, and in average they use 2-3 words. This affects the quality of the ranking but it simplifies the query algorithm.
- 15% restrict the search to a certain subject.
- 80% does not modify the query (feedback zero).
- 65% of the queries are unique.

Next, different methodologies for the crawling-indexer-searcher modules are proposed and analized, according to the different search strategies presented in [5, 6]. This paper is focused in the index construction task.

## 7.1 Local Strategy

The local index strategy is very simple because the inverted list is built based on the documents that each processor has. Here, each processor builds its inverted list using its own local documents, therefore each machine will contain a table with the same $T$ terms, but the length of the identifiers list of the associate documents is approximately $1/P$, where $P$ is the number of the server's machines.

For this strategy, the indexer maintains a queue with URL of the documents recovered by the crawler where each documents referenced by the URL are processed one at the time, due how the server's processors work (each processor builds its own inverted list using its local documents).

So in this strategy, the indexer's task is to select the server's processor which will receive the terms of a document and then the selected processor will insert the terms with their respective associated lists in the local inverted list.

The indexer must process the documents to extract the relevant terms with its additional information necessary to perform the ranking operation (document identifier, frequencies, etc.), as it were explained in the Section 5, in order to create a temporary inverted list that will be sent to the selected processor, which will have to update its local inverted list with this new information. As consequence, this processing implies a high degree of communication between the processors of the BSP server and the indexer.

The processors may be selected at random or using a hash function, but to control that all the processors maintain approximately the same amount of documents processed, the indexer may have a counter for each machine to maintain the number of documents processed in each one. Therefore at the moment to send the terms of a new document entered to the system, the indexer will select the processor whose counter has the smallest value.

The index update can be easily performed in parallel if the indexer is formed by a set of machines connected by a network, and the crawler is oriented to batch, that is to say, the crawler distributes a number of URLs between the machines of the indexer at random. On the other hand, the query processing consists on routing the query to a processor, broadcasting this query (first superstep of the server), then retrieving the associated inverted lists to be sent to the ranker machine (second superstep), and finally performing the ranking of documents to be presented to the user (third superstep).

## 7.2 Global Strategy

The global index approach consists, contrary to the previous one, on distributing uniformly at random (hashing) every vocabulary word and its associated list among the processors. It is to say that the whole documents collection is used to build a vocabulary table that is identical to the sequential one. Then $T$ terms that form the global table of terms, are uniformly distributed on $P$ processors along with their respective identifiers lists. Therefore, after mapping the terms among the different processors, each one contains approximately $T/P$ terms.

In this case, the indexer also maintains a queue with the URLs of documents recovered by the crawler, but unlike the previous strategy, the indexer processes several documents to construct the temporary inverted list before distributing the terms with its corresponding associated lists, in order to be able to reduce the communications between the server's processors and the indexer.

The selection of the processor that receives the terms is made by a hash function, that is the same one used by the broker machine to distribute the terms of the queries that arrive from the users machines connected to Internet. This hash function considers the term or word and the number of processors that conform the BSP server.

This indexer also can be paralleled as in the previous case, because the distribution task of the URLs carried out by the crawler, is independent of how the indexer performs the document processing and how this one distributes the temporary inverted list.

The parallel queries processing consists on determining to what processors to route every word that compose the query (this is performed by the broker machine) and then retrieving the associated inverted lists performing a preranking (first superstep) and lastly performing the ranking of documents to be presented to the user (second superstep).

## 7.3 Bucket Strategy

Two kinds of buckets strategies are presented. First, buckets distributed among different supersteps, and then buckets distributed among different processors with circulate, sequential, hash and random distribution. The goal of these strategies is to reduce the size of the associated lists recovered from secondary memory, and the time required to process the queries.

### 7.3.1 Bucket Distributed among Different Supersteps

In order to perform the construction and update of the inverted lists, this strategy behaves just as the global one. Here, the main idea is to divide the associated lists of each term in *buckets* of size $K$, keeping only one of them in main memory, the one with the higher frequencies, while the others remain in secondary memory, but each processor has the terms with the whole associated list. It is very important to get an optimal $K$ such that the processors don't need to access many times to secondary memory, and in the other case the lists don't occupy too much memory.

In this case there are two extreme cases, the worst one is when a processor has to search in all the *buckets* of a term, because it has higher frequencies than any other processor. And the best case is when all processors only need the first *bucket* to solve the query.

The queries processing under the *BSP* model is as follow:

1. First the processors get the queries, process them and send them to the *ranker*.

2. Then the *rankers* machines will get the messages and will check if some processor has a *bucket* with higher frequencies than the received. If that doesn't happend, the *rankers* will perform the final ranking and will send the best documents identifiers to the broker machine. In the other case, if more *buckets* are needed, the *rankers* will send a message with the terms to the processor that has a *bucket* with higher frequencies.

3. This superstep is executed if more *buckets* were requested in the previous superstep. Here, the processors get the messages with the terms that require the next *bucket*, recover them and send them again to the *ranker*. Then they return to the second superstep.

### 7.3.2 Bucket Distributed among Different Processors with Circulate and Sequential Distribution

These distributions combine the global index strategy for the inverted lists building, and the local index strategy for the queries processing. To build the vocabulary table with the relevant terms and their corresponding associated lists, the complete collection of documents of the textual database must be considered. But contrary to the global strategy, the associated lists are divided in buckets and then these buckets are distributed among the processors. The associated list consists of pairs <d, $f_{d,t}$>, where $d$ is the document identifier and $f_{d,t}$ is the frequency of the term $t$ in the document $d$.

In the sequential distribution, the associated lists of each term are divided in buckets of size $K$. If $N$ is the number of pairs <d, $f_{d,t}$ > for a term, then $K = N/P$ , where $P$ is the number of processors. Here, each processor will receive approximately the same quantity of pairs <document-frecuency> for each term. Then these buckets are distributed among the different processors, so that the bucket i of the term t is sent to the processor i, it is to say that bucket 0 goes to the processor with logic identifier 0, and so on. An unwanted characteristic of this distribution, is a loss of load balancing among server's processors.

The circulate distribution groups the pairs <documents-frequency> in *buckets* of size $K$, but contrary to the previous distributions, the *buckets* are distributed among the processors in a circulate way as indicates its name. It is to say that the *buckets* of the $term_1$ are distributed following the sequence $P_0, P_1, P_2,..,P_{P-1}$, then the *buckets* of the $term_2$ are distributed following the next sequence $P_1, P_2,..,P_{P-1}, P_0$ and so on.

Due to the use of bucket, the task of the indexer unlike the global strategy, requires additional information for the associate lists buckets distribution. This information talks about an identifier of bucket which is used by the hash function that selects the processor that will update its inverted list. So this hash function now takes the term or word of the temporary inverted list, with the number of the server's processors and the bucket identifier.

Consequently, once the indexer performs the parsing of documents recovered through the URLs, and obtains the frequencies along with the documents identifiers, it has to divide by each term the associated lists in buckets of size $K$ (previously established), assigning it its bucket identifier.

### 7.3.3 Bucket Distributed among Different Processors with Hash and Random Distribution

Up to now, the presented strategies for the distribution of the vocabulary table's terms, work with *buckets* of fixed size $K$, which is calculated considering the number of processors $P$ and the quantity of the associated pairs $< d, f_{d,t} >$ to each term. In this section, a hash and a random distribution which allow to obtain the processors identifiers to which the respective *buckets* will be sent, is shown.

The inverted list construction, requires to build the vocabulary table with the relevant terms and its respective locations in decreasing order according to the frequencies calculated for each document in which the term appears. Then, the inverted lists should be divided in *buckets* of variable size $K$ in a range of $2, .., N-1$, where $N$ is the number of pairs that each term has (the cases $K = 1$ and $K = N$ are avoided).

To distribute the *buckets* among the server's processors, a *hash* function that considers the term (because some terms have higher probability of appearing than others), the identifiers number of the *bucket*, (so that not all *buckets* go to the same processor) and the number of processors should be used. If $K$ is big, then the number of *buckets* is small, the data distribution over the different processors is poor, and the concurrence during the queries processing is bigger. But a small $K$ allows a good distribution of the data and a bigger parallelism during the queries processing.

A difference presented between the hash and random distribution, is that the second one uses an additional srtucture to know what processor contains buckets for a particular term, therefore it has to perform another operation during the updating phase. It must warn to the broker that new information has arrived to the system and what processors have this information; therefore broker will know, at processing time, to which processor send the arriving queries.

This hash function, used by the index when it updates the lists, reduces the probability that one processor receives more than one *bucket* with high frequencies. The queries processing is similar to the previous strategies, but due to the use of the *hash* function to distribute the *buckets*, now the *broker* machine has to carry out an additional control before sending the queries generated by the user to the server. This control implies to identify the processors that contain *buckets* for the terms that appear in the query. Once the processors are identified, the *broker* generates a sub-query for each one of these processors with the terms that correspond to them.

When the processors receive the sub-query (first superstep), they will recover the list for this sub-query and will send it to the *ranker* machine. This last one (second superstep), receives a partial results and will make the final ranking to select the best documents.

## 8 Conclusions and Future Works

In this paper, the existing relation between the searcher-indexer-crawler modules of a search engine in the Web is presented. For this, the task that each one of them perform is described and the job that these modules perform is shown according to the different indexing strategies that have been studied in previous works.

A parallel approach under the BSP model to implement the search strategies is presented in this paper, and the indexer job, which plays an important role in a search engine at the time of creating the index and on which generally is not discussed, is analyzed.

The indexer is the one which builds and updates the inverted lists of the BSP server, which has assigned the documents processing task which requires greater time and resources of CPU, with the purpose of reducing the server load work that the server's processors have during the queries processing.

This work is the beginning of a deep investigation that will be carried out to analyze strategies of pages recovery to improve the performance of crawler as well as strategies of queries processing with the purpose of speed up the response times to the users requests.

As future work is intended to investigate the amount of documents to be processed by the indexer, is to say if it should process all referenced documents by the URLs that it has in its waiting queue or if it must process them by batchs. In this last case the batch size should be analyzed. Both cases are related to the crawler's job batch oriented. Also the optimal size of the batchs of documents maintained in the queue of URLs by the crawler will be analyzed.

## References

[1] R.Baeza-Yates,B.Ribeiro-Neto. “Modern Information Retrieval”. Addison-Wesley-Longman 1999.

[2] K. Bharat and A.Z. Broder. “A technique for measuring the relative size and overlap of public web search engines”. In Proceedings of the 7th World Wide Web Conference, pages 379-388, 1998.

[3] M. Burner. “Crawling towards eternity: Building an archive of the world wide web”. In Web Techniques, 1.

[4] Cho. Garcia-Molina. “The evolution of the web and implications for an incremental crawler”. In Proceedings of the 26th International Conference on Very Large Databases, 2000.

[5] G. V. Gil Costa. “Procesamiento Paralelo de Queries sobre Base de Datos Textuales”. Tesis de licenciatura. Universidad Nacional de San Luis. 2003.

[6] Veronica Gil Costa, A. Marcela Printista. “Estrategia de Buckets para Listas Invertidas Paralelas”. XII Jornadas Chilenas de computación. Arica, Chile. 8-12 de noviembre del 2004.

[7] V. Gil Costa, M. Printista y M. Marín. “Algoritmos para Listas Invertidas Paralelas”. WICC 2004, pág. 545-548. Mayo 2004, Argentina.

[8] V. Gil Costa, M. Printista y M. Marín. “Modelización de Listas Invertidas Paralelas”. X Congreso Argentino de Ciencias de la Computación, 4-8 de Octubre 2004.(CACIC 2004).

[9] Google. http://www.google.com.

[10] M. Marín. “Parallel text query processing using Composite Inverted Lists”. In Second Intenational Conference on Hybrid Intelligent Systems (Web Computing Session). IO Press, Feb. 2003.

[11] M. Marin, C. Bonacic y S. Casas. “Analysis of two indexing structures for text databases”, Actas del VIII Congreso Argentino de Ciencias de la Computación (CACIC2002). Buenos Aires, Argentina, Octubre 15 - 19, 2002.

[12] Maxim Martinov, Boris Novikov. “An Indexing Algorithm for Text Retrieval”.Proceedings of the International Workshop on Advances in Databases and Infotmation Systems, (ADBiS’96). Moscow, September 10-13, 1996.)

[13] M. Printista, V. Gil Costa y M. Marín. “Búsquedas en Base de Datos Distribuidas en una red de Procesadores”. IX Jornadas Iberoamericanas de Informática, Cartagena de Indias, Colombia. 11-15 de Agosto, 2003. ISBN:84-9602307-9.

[14] Sergey Brin, Lawrence Page. “Google: The Anatomy of a Large-Scale Hypertextual Web Search Engine”. Proceedings of the 7th International World Wide Web Conference, pages 107-117, April 1998.

[15] Wilkinson B. And Allen M. “Parallel Programming Techniques and Applications using Networked Workstations and Parallel Computers”. Prentice Hall -1999.

[16] L. Valiant. “A Bridging Model for Parallel Computation”. Communications of the ACM, Vol. 33, Pp 103-111, 1990.

[17] A. MacParlane, J.A.McCann y S.E. Robertson. “Parallel Search Using Inverted Files”. In the 7th. International Symposium on String Processing and Information Retrieval, 2000.

[18] 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.

[19] WWW.BSP and Worldwilde Standard, http://www.bsp-worldwide.org

[20] WWW.BSP PUB Library ar Paderborn Univertity, http://www.uni-paderborn.de/bsp