Buckets Inverted Lists for a Search Engine with BSP

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

Most information in science, engineering and business has been recorded in form of text. This information can be found online in the World-Wide-Web. One of the major tools to support information access are the search engines which usually use information retrieval techniques to rank Web pages based on a simple query and an index structure like the inverted lists. The retrieval models are the basis for the algorithms that score and rank the Web pages. The focus of this presentation is to show some inverted lists alternatives, based on buckets, for an information retrieval system. The main interest is how query performance is effected by the index organization on a cluster of PCs. The server design is effected on top of the parallel computing model *Bulk Synchronous Parallel-BSP*.

Keywords: Search Engine, Buckets, BSP, Textual Databases, Supersteps.

1. INTRODUCTION

One of the major tools for information access are the search engines. 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, the new search engines must deal with information that is much more heterogeneous, messy, more varied in quality, and vastly more distributed or linked. In the 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).

In response to this huge expansion of potential information sources, today's web search engines have emphasized speed, with less importance attached to effectiveness. Because of this, several studies have been met to the development of new strategies that allow to satisfy this demands through the parallel processing, that has demonstrated to be a paradigm that allows to improve the algorithms execution time.

For efficient query processing, specialized indexing techniques have to be used with large documents 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 [19]. Each of them has 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 document collection is indexed using inverted lists.

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 seraching 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 (vectorial model [4]).

In the following sections, the bucket strategies that implement the parallel inverted lists, which attempt to reduce the execution time required to process the queries coming from different users, are analyzed through the BSP model.

2. PREVIOUS WORK

In previous works, parallel algorithms for the local inverted list and global inverted list strategies have been developed, using an analysis and well structured design methodology, through the BSP computing model [7, 9]. Paralellization using the global index approach consists on distributing uniformly at random every vocabulary word and its associated list accross the processors. Thus processing a query in parallel consists on determining to what processors

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

route every word that compose the query, and then retrieving the associated lists to perform the ranking of documents that will be presented to the user.
The local index case is very simple, the inverted list is built using the documents that each processor has. Here, each processor builds its inverted list using its own local documents, therefore each machine will have a table with the same T terms, but the length of the associated list with the document identifiers will be approximately $1/P$, where P is the number of server's machines; and the queries processing operation consists to route the query to a processor, then broadcasting this query, to then retrieve the associated lists and to finally perform the ranking of documents.
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 [11] [15]. These experiments have proven that this structure can be efficiently processed in parallel.
Also the query performance have been studied to analyze how it is affected by the network speed, and the disk transfer rate under these index organization [14].
The outcome of this works is as follows. First the computing model and the architecture used to do this work are shown and then the proposed strategies. Preliminary versions of this work appear in [7, 8].

3. COMPUTING MODEL

In the *Bulk Synchronous Parallel, BSP* model of computing, proposed in 1990 by Leslie Valiant [17], 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 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 messages are available for processing at their destination by the next superstep, and each superstep is ended with a barrier synchronization of processors [16]. 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 $BSPlib$ [19] or $BSPpub$ [20]. 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 and MPI libraries.
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, $h * G$ y L, where w is the maximum number of calculation performed by each processor, 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 synchronising 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 8 *SMP* (dual) workstations connected by fast switching technology. A network of workstations is an attractive alternative nowadays due to the emergent fast switching technology provides fast message exchanges and consequently less parallelism overhead. In this network, each machine has its own operating system, and the communication is made by a messages passage library.
To process the user queries, 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 a *target* machine of the server.
Also, for each received query, one of those P server's processors will be the *ranker*, which is selected by the $broker$ during the queries distribution time. This $ranker$ will perform the final ranking of document and will send them to the requesting user machine.

5. BUCKETS DISTRIBUTED AMONG DIFFERENT PROCESSORS

This strategy proposes to group the associated lists in buckets, and then distribute these buckets among the different processors (BADP). Its goal is to reduce the processing time and the data size that has to be recovered from secondary memory.
Four distribution are presented for this strategy: uniform sequential and circular distribution, a hash distribution, and lastly a random distribution.

5.1. Uniform Sequential and Circular 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 associated lists, the complete collection of documents from the textual database must be considered. The associate 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. The pairs of the associa-

ted lists are in decrease order by their frequencies. Then the associated lists are divided in buckets of size $K = N/P$, where P is the number of processors in the BSP server, and N is the number of pairs $< d, f_{d,t} >$. So, in this way, the first buckets will have higher frequencies than the last ones.
In the uniform sequential distribution, these buckets are distributed among the processors in a sequential way, so the $bucket_0$ goes to P_0, the $bucket_1$ goes to P_1, and so on. As you can observe, the processors with low logical identifier receive buckets with higher frequencies, and processors with high logical identifier receive buckets with lower frequencies.
On the other hand, in the uniform circular distribution, the buckets of the terms are distributed among all processors in a circular way as indicates its name. So, 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 sequence P_1, P_2,.., P_{P-1}, P_0 and so on, like it is shown in the Figure 1.

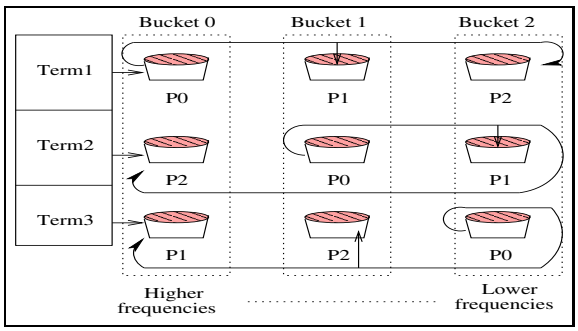

Figure 1: Uniform circular distribution for a server with three processors

The queries may have one or more terms and some of the letters (a,c,m,p) that are in these terms have bigger probability of appearing than others. The broker machine has to receive the queries and has to send them to the server. To make this, it will select a ranker and a target machine for each query.
The BSP model can predict the query processing operation cost, and for that we assume that the associated lists are stored on secondary memory. Also, it is considered the query processing cost since the broker machine sends the queries until this machine receives the results from the BSP server [12]. The secondary memory is treated as the network communication. It is to say, that a parameter D is included to represent the average cost of accessing the secondary memory. This parameter can be easily obtained using benchmark programs from the Unix systems. If the database index can be completely stored in the P main memory, then $D = 1$.
The execution of a lot of $Q = qP$ queries using the sequential distribution, is as follows. In the first superstep, the processors get q queries and broadcast

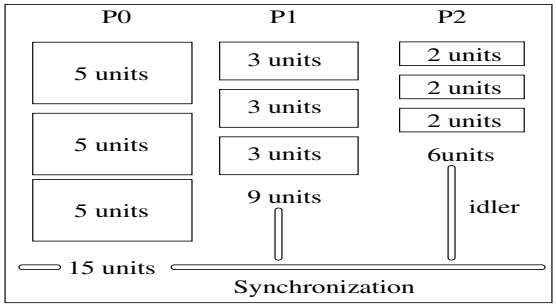

Figure 2: Processing time for the sequential distribution

them with a cost of qP, a synchronization barrier ends this superstep. So this superstep has a cost of $t_1 = q + qPG + L$, where L is the synchronization cost, qP is the h-relationship (the maximum number of messages send/received in this superstep), and G is the cost in words of sending the message.
In the second superstep, the processors will get the $qP = Q$ queries, then they will join the queries coming from the different processors, and they will work on the determination of the document identifier lists, using the vectorial model, for each one of these queries, to built the partial result with a cost of $KD\gamma_{id}$, where $K = N/P$ is the bucket size (N is the number of pairs $< d, f_{dt} >$), and $\gamma_{id} >= 1$ is the load factor (see Eq.(1)) that reflects the workload of the processors. So in this superstep the cost is $t_2 = Q + qDK\gamma_{id} + qKG + L$, where qKG is the cost of sending the partial result to the ranker, and in the worst case they will send K partial result. It is important to see that the processors with low identifiers receives the buckets with higher frequencies. If the $processor_{id}$ has a low logical identifier, γ_{id} will return a high value, in other case, γ_{id} will return a value close to one. Therefore, the load factor γ allows to represent the work disparity that each processor has.

$$\gamma_{id} = \rceil K * ((B - bucket_{id})/B) \qquad (1)$$

The parameter B is the number of buckets for a term, and in this distribution, B is equal to P when the number of pair <document,frequency> is bigger or equal to the number of processors.
If the server has three processors, where P_0 requires 5 units of time to process each bucket, P_1 requires 3 units of time and P_2 requires only 2 units of time, as its shown in the Figure 2, then due to the BSP model synchronize all processors to the higher time spent by any processor, the tree processors of this server will synchronize at P_0 time, making the others ones remain idlers 6 and 9 units of time.
Finally, in the last superstep, the ranker processors receive qK messages from the others to perform the final ranking with a cost of qK and send the results to the broker machine. Therefore the cost of this superstep is $t_3 = qKP + qK + qKG + L$. Then the

asymptotic cost of this distribution is the sum:

$$\sum_{s=1}^{3} t_s = q(1+P+K(1+P+\gamma_i D))+q(K+P)G+L \quad (2)$$

In the first superstep, this distribution consumes a lot of communication and synchronization time, because of the broadcast. This cost will grow up as the number of processors is increased. The second and third superstep, have more computation and the communication depends on the query being processing.

The disadvantage presented by the sequential distribution, is that the documents with higher weights will fall among the first logical processors (P_0,P_1,..), making a work overload over these. In the circular distribution the buckets with the highest frequencies will not always go to the same processor, and due to this the unbalance of workload presented in the sequential distribution can be compensated. Because of this, the cost of the circular algorithm using the BSP model, for the execution of a lot of $Q = qP$ queries, is just like the described before but without γ, that represents the load factor, because in this case the average time required by each processor to process the queries is the same (see Figure 3).

An optimization that has been applied to these search strategies is the use of filters proposed in [13] which allows to filtrate documents during the ranking operation with a significant reduction in the ranking evaluation cost without degradation in retrieval effectiveness. The filtering method considers as candidate answers only the documents that with high within-document frequency. The memory usage is reduced because having fewer candidates means that fewer accumulators are required to store information about these candidates. Disk traffic and CPU processing time are also reduced, because by ordering inverted lists by decreasing within-document frequency, only the first portion of each list containing high frequencies will be processed, and the rest can be ignored.

Although, this filtrate technique does not benefit all the distributions proposed, because the uniform sequential distribution will be harmed. That is because the processors with low logical identifiers have documents with higher frequencies than the processors with high logical identifiers, leaving these last ones idlers at the queries processing time.

Another important issue in a Web engine, is the space required to store the index. If we are working with a vocabulary table of size T (that means T terms), and the number of pairs in the associated lists is approximately N, then the space needed to store the index in each processor is shown in Eq.(3), where $k = \rceil N/P$ is the bucket size (that is to say, the number of pairs in each bucket), and in these distributions every machine gets only one bucket.

$$T * \rceil N/P \quad (3)$$

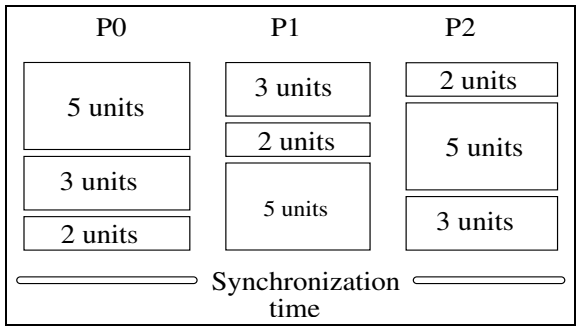

Figure 3: Processing time for the circular distribution

So, the processors will have approximately the same term, but the associated list will be smaller.

5.2. Hash and Random Distribution

Up to now, the presented 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 number of pairs $< d, f_{d,t} >$.

In these cases, the global organization is used for the inverted list construction and it requires building the vocabulary table with the relevant terms and its respective locations in decreasing order by their frequencies. Then, the associated lists is 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). If K is big enough, 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 data distribution and a bigger parallelism during the queries processing.

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 not all the buckets go to the same processor) and the number of processors is be used.

This hash function reduces the probability that one processor receives more than one bucket with high frequencies. The queries processing requires just two supersteps, and due to the use of a hash function, the broker machine has to perform an additional control before sending the queries to the server. This control implies to identify the processors that have buckets for the terms that appear in the query. Once the processors are identified, the broker generates a subquery for each one of these processors.

In the random distribution, the processors that receive the buckets of the inverted lists are randomly selected. Due to this, it is necessary that as the buckets are distributed among the processors, the broker machine has to update a structure with the following format:

$< term_1, < P_1, .., P_n >>$ where the second list corresponds to the processors that have buckets for that term. As in the previous case, the broker machine has to select the processors that have buckets for the terms of the received queries, but using the structure mentioned before. Then it has to generate a sub-query for each processor that has a bucket for the terms of the query, exploding the $bulk$ property of BSP [10].
In both distributions, when the processors receive the q sub-queries, they will search in their inverted list for the terms of these sub-queries with a cost of qDK, and will send the found document identifiers as partial result to the ranker machine (qKG) with a cost of $t_1 = q + qDK + qKG + L$.
In the second superstep, the ranker processors will get qK messages with the partial result, and will perform the final ranking with a cost of qK. Finally, they will send a list with the top document identifiers to the broker (qKG). Therefore, this superstep has a cost of $t_2 = qKP + qK + qKG + L$. So, the asymptotic cost of this distribution is:

$$\sum_{s=1}^{2} t_s = q(1 + K + P) + qKG + L \qquad (4)$$

So these distributions require only two supersteps to perform the queries processing, and they allow more concurrency. The only difference between both, is that the random requires an additional structure to know which processor has information about a term. So the queries processing and the analytic cost is the same, and it is not expected to have a significant variation of values in the empirical experiments.
The purpose of using a random distribution is to be able to measure how good is the selected hash function. These two last distributions reduce the probability that one processor receives more than one bucket belonging to the same term, and when the buckets are bigger enough a better load balance during the queries processing can be obtained.
Finally, analyzing the space metric, and due to these distributions have different bucket size; if T is the number of term in the vocabulary table, N is approximately the associated list size, and P is the number of processors in the server, then we have three possible cases. In the first one if the number of processors P is equal to the number of buckets $B = \rceil N/K$, then the space required is shown in Eq.(5). Here, each processor will have one bucket for each term of size K. In the second case, if if $B < P$ then the Eq.(6) shows the space required. In this case, some processors will not get buckets for some terms (T/P), and the buckets will have a K size. Finally, the space needed if $B < P$, is shown by Eq.(7), where again every processor will get the T terms and some of them may get $\rceil B/P$ buckets of size K for these terms.

$$T * \rceil N/B = T * K \qquad (5)$$

$$T/P * \rceil N/B = T/P * K \qquad (6)$$

$$T * [\rceil N/B *] B/P] = T * [K *] B/P] \qquad (7)$$

6. BUCKETS DISTRIBUTED AMONG DIFFERENT SUPERSTEPS

This strategy distributes the buckets among the supersteps of a processor (BADSS) according to the BSP model. The main idea is to divide the associated lists in buckets of size K, keeping only one of them in main memory, the one with the higher frequencies.
The construction of the inverted list is just like in the global strategy, where a sequential vocabulary table is built and then the terms are distributed among the processors with their whole associated lists.
Therefore it's remains to explain how to perform the queries processing (see Figure 4). First all processors have to recover from secondary memory the first bucket of each term and the frequency of the next bucket. Once all processors are ready they can begin with the first superstep, where some of them, previously selected by the broker machine, receive the queries and select the top documents (forming a partial result) for these queries. These documents are send to the ranker machine and then there is a barrier synchronization, which separates the supersteps. After this synchronization, the ranker machines proceed as follows:

```
1: procedure
2:     for each query do
3:         switch(message.type)
4:         case ranking:Rank()
5:         case new_bucket:Get the results from the
6:         next bucket and send them to the ranker
7:     end for
8: end procedure
```

In the $Rank()$ function the ranker has to check if the partial results received are new buckets previously required for a term. If that so, it has to replace the old buckets with these new ones. Then for each term of the query been processing, the ranker has to verify if some processor has a bucket with higher frequencies than the received. If that happens, then it has to request a new bucket to those processors and return. In other case, the ranker performs the final ranking operation to return the top documents.
With this strategy 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. And the best case is when all the processors only need the first bucket to solve the query.

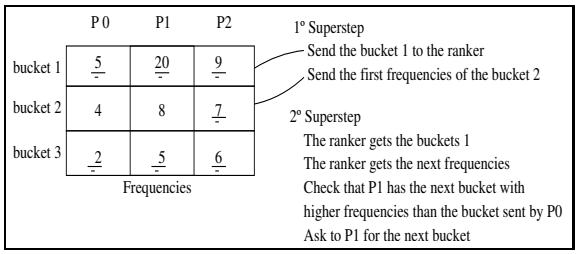

Figure 4: Bucket distributed among different supersteps

The cost of this strategy under the BSP model assuming T the maximum number of terms that will be required by the next superstep, is as follow:

1. First the processors get q queries, process them and send the partial results to the ranker with a cost of $q + qDK + qKG + L$.

2. Then the ranker machines will get qK messages and will check if some processor has a bucket with higher frequencies than the received. If that does not happens, the rankers will perform the final ranking and will send the top document identifiers to the broker, with a cost of $qK + qK + qKG + L$. In the other case, if more buckets are needed, the rankers will send a message with the terms to the processors that have buckets with higher frequencies. So the cost of this superstep would be $qK + TG + L$.

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 from secondary memory, select the top document identifiers and send them again to the ranker with a cost of $T + TDK + TKG + L$. Then return to the second superstep.

A variant for this strategy will be that the target machines, that generate the partial results, send only K/P results instead of K (KBADSS). This allows reducing the communication between the processors and the time worn-out in the synchronization barrier. As with the previous strategy, now the size metric will be show. In this case, the vocabulary table has T terms and due to each term is send to a processors with the whole associated list, each processor will need the space shown by the Eq.(8).

$$T/P * N \tag{8}$$

7. EXPERIMENTAL RESULTS

In this section the experiments performed using a 2GB sample of the Chilean Web with a query log from www.todocl.cl, the Fibrosis database (4.7 Mb) with 1239 documents published from 1974 to 1979 [14] and the Magazine database (1.5 Mb) with 546 documents of (ftp://ftp.cs.cornell.edu/pub/smart/time/) will be shown. 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 local index and the global index organization. The experiments were performed with a filter Cins=0.12 [13] and Cadd=$\{0.0,0.25,0.50,0.75,1.0\}$. The developments were performed in a cluster of 8 SMP (dual), connected by a $FastEthernet$.

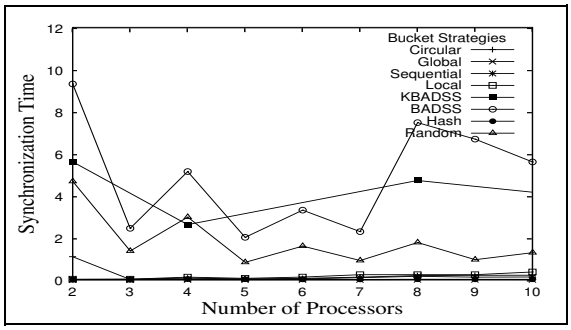

Figure 5: Running Time - Chilean Database.

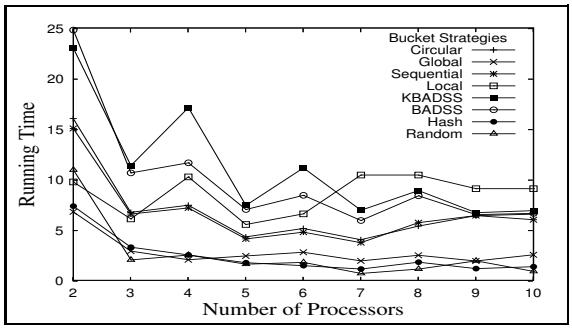

Figure 6: Running Time - Fibrosis Database.

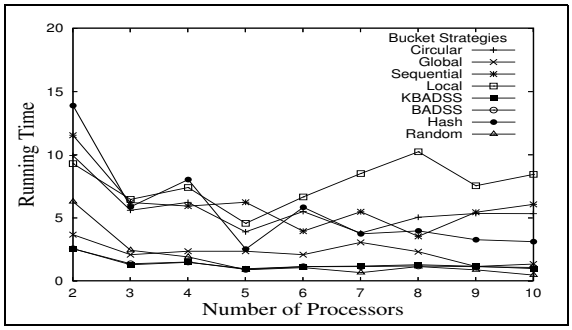

Figure 7: Running Time - Magazine Database.

Figure 5 shows the running time for the execution of a lot of Q=2000 queries under the Chilean database. Figure 6 shows the times obtained by each strategy for a lot of Q=100 queries with the Fibrosis database; and finally the Figure 7 shows the running time obtained for a lot of Q=81 queries. With the first two databases, where the associated lists are not bigger than the number of terms in the vocabulary table, the hash and random distribution for the BADP strategy, followed by the global organization, perform better that the others. But in the Magazine database where the associated lists are really big, the BADSS and the KBADSS outperforms the others, because these ones work with smaller lists and in generally they do not require more than one superstep to complete a query request. In all the cases as the number of processors is increased, the speedup in the query processing operation is limited, at the bottom, by the communication and synchronization time, and the network traffic.

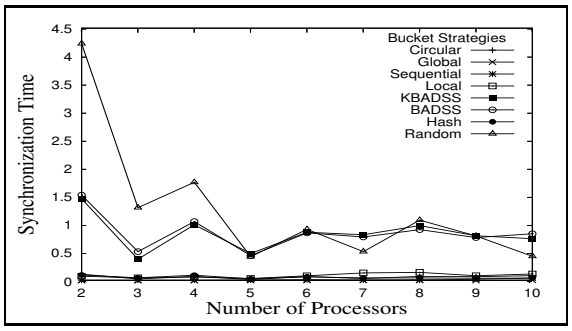

Figure 8: Synchronization Time - Magazine Database.

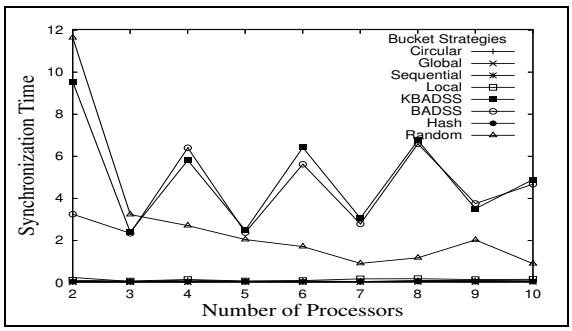

Figure 9: Synchronization Time - Fibrosis Database.

Then Figures 9,8 and 10 show the synchronization time for each one of the presented strategies and the global and local organization, using the three databases. In the three of them, the BADSS and KBADSS consume more communication than the others ones, allowing confirming the behaviour presented in the running time figures. That is to say, the experimental results obtained by these two strategies are dependent on the communication and synchronization time consumed by each one.

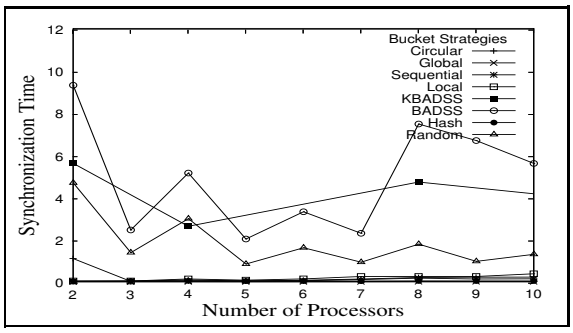

Figure 10: Synchronization Time - Chilean Database.

The bucket size is also studied in this work, trying to find the optimal size for each one of these strategies. But as the Figures 11 and 12 show, as the number of buckets is increased the running time does not varies very much, that is because as the number of buckets grows up the computation and the communication also does.

Another important issue is that when the number of pairs in the associated lists is smaller then the number of processors ($N << P$) there is a moment when the computation cost can not be reduced any more and the running time arrives to a threshold that can not pass. The ideal case is when the number of buckets is equal to the number of processors ($N \, mod \, P = 0$), so every machine gets only one bucket, but when $N mod P <> 0$ some processors will have more buckets requiring more computing time than others.

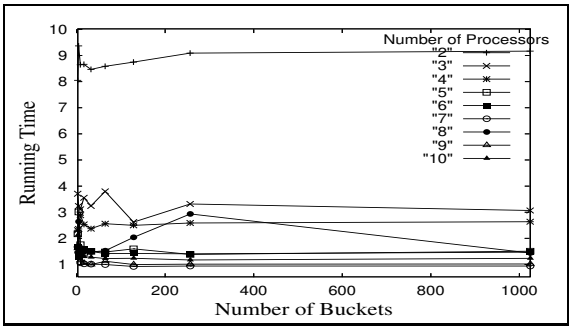

Figure 11: Bucket size with the hash distribution - Fibrosis database.

8. CONCLUSIONS

The query processing performance has been studied with strategies that use the concept of buckets for the parallel implementation of the inverted lists index. Inverted lists are used as index structures and the vector model is adopted as ranking strategy.

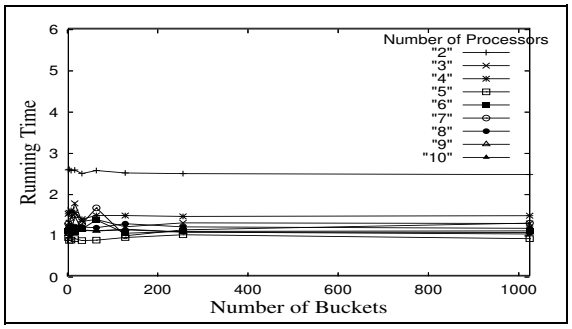

Figure 12: Bucket size with the BADSS strategy - Magazine database.

The proposed strategies are compared with the local and global index strategies. The goal of the presented strategies is to divide the associated lists in buckets of size K, and to distribute them among the server's processors, to be able to reduce on one hand the storage space required by these lists in each processor, and the processing time.

The study of these strategies is based on the BSP model. The theorical analysis shows that the BDASS requires a lot of communication and synchronization, if the ranker has to ask documents from more than one bucket, and the number of supersteps depends on the query and the frequencies that each processor has.

Also the circular and sequential distribution analysis show that the cost of the broadcast is high. Finally, the random and hash distribution requires the smallest quantity of supersteps (only two), and they allow greater concurrency among the various queries and less communication.

The results for the Chilean Web and the Fibrosis databases indicate that for small databases the global strategy and the hash distribution get similar running times, and the times obtained by the others are above these ones. But for big databases, where the use of parallelism is justify, the buckets strategies outperforms the local index organization and get better running times than the global index organization.

As future work it is intended to work with another type of data structure, like the trees (SAT,dSAT,etc.), that also allow to search texts and another multimedia information like sounds, videos, etc.

References

[1] Serge Abiteboul and Victor Vianu. "Queries and Computation on the Web". Proceedings of the International Congerence on Database Theory. Delphi, Greece 1997.

[2] C. S. Badue. "Distributed query processing using partitioned inverted files". Master's thesis, Federal University of Minas Gerais, Belo Horizonte, Minas Gerais, Brazil, March 2001.

[3] R. A. Barbosa. "Departameho de consultas em bibliotecas digitais fortemente aclopadas". Master's thesis, Federal Univerity of Minas Gerais, Belo Horizonte, Minas Gerais Brazil, May 1998. ——-in Portuguese.

[4] R. Baeza and B. Ribeiro. "Modern Information Retrieval". Addison-Wesley. 1999.

[5] R. Baeza-Yates and A. Moffat and G. Navarro. "Searching Large Text Collections", Handbook of Massive Data Sets,Kluwer Academic Publishers, 2002, ISBN 1-4020-0489-3.

[6] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Nielsen, and A. Secret."The World-Wide Web". Comm. of the ACM, 37(8):76-82,aug 1994.

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

[8] 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.

[9] 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).

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

[11] 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.

[12] 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.

[13] M. Persin, J. Zobel, R. Sacks-Davis. "Filteres Document Retrieval with Frequency-Stores Indexes". Journal of the American Society for Information Science, 1996.

[14] B.A. Ribeiro-Neto and R.A. Barbosa. "Query performance for tightly coupled distributed digital libraries". In Third ACM Conference on Digital Libraries, pages 182-190, 1998.

[15] 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.

[16] D.B. Skillcorn and J. Hill and W.F. McColl. "Questions and Answers about BSP". Oxford University Computing Laboratory. PRG-TR-15-96. 1996.

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

[18] I. Qitten, A. Moffat and T. C. Bell. "Managing Gigabytes - Compressing and Indexing Documents and Images". Morgan Kaufmann Publishers, Inc. second edition, 1999.

[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

Received: Nov 2005. Accepted: Feb. 2006.