# Multimedia Web Searches using Static SAT

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

and

**Mauricio Marín**
Computing Department University of Magallanes
Punta Arenas, Chile
marin_umag@yahoo.com

**Abstract**

In this paper we describe the parallelization of a data structure used to perform multimedia web searches. Multimedia Web Engines have not been deeply studied and is a challenging issue. The data structure selected to index the queries is the Spatial Approximation Tree, where the complexity measure is given by the number of distance computed to retrieve those objects close enough to the query. We present a parallel method for load balancing the work performed by the processors. The method can adapt itself to the changes of the workload produced by the user queries. Empirical results with different kind of databases show efficient performance in a real cluster of PC. The algorithm is designed with the bulk-synchronous model of parallel computing.

**Keywords:** Metrics Space, BSP, Web Engine, Spatial Approximation Tree, Indexes.

## 1 INTRODUCTION

The plentiful content of the World Wide Web is useful to millions. Some simple browse the Web through entry points such as Yahoo!. But many information seekers use a search engine to begin their Web activity. In this last case usually users submit a query of keywords, and receive a list of Web pages containing the keywords that may be relevant.

There is no question that the Web is a huge and challenging to deal with. Several studies have estimated the size of the Web [4, 1], and while they report slightly different numbers, most of them agree that over a billion pages are available. Given that the average size of a Web page is around 5-10K bytes, just the textual amounts to at least tend of terabytes. The growth rate of the Web is even more dramatic, it is projected to be doubled every two years [22].

For helping users to find their answers, the search engine module is responsible for receiving and filling search request from users. The engine relies heavily on the indexes and some time on the page repository. Because of the Web's size, and the fact that users typically only enter one or two keywords, result sets are usually very large. Therefore the ranking operation has the task of sorting the results. Results near the top are the most likely ones to be what the users are looking for. The query module is of special interest because traditional information

retrieval (IR) techniques have run into selectivity problems when applied without modifications to Web searching: most traditional techniques rely on measuring the similarity of query texts with texts in a collection's documents.

So far, we have assumed that web repository store only plain text or HTML pages. However with the growth of non-text content of the Web, it is becoming increasingly important to store, index and search over images, audio, and video collections.

Very little research work has been done on this area in the context of Web engines. But there are a lot of data structure studies applied over this issue, some of them are BKTree [7], GNAT [6], M-Tree [10], etc. These structures are used to perform similarity searches in metric spaces. A metric space is formed by a collection of objects $U$ and distance function $d$ defined among them, which satisfies the triangle inequality. The goal is given a set of objects and a query, to retrieve those objects close enough to the query.

A recent data structure is the Spatial Approximation Tree (SAT) devised to support efficient searching in high-dimensional metric spaces [19] to perform multimedia Web searches. This structure has been compared successfully against other data structures [8, 12] and update operations have been included in the original design [21, 20, 11].

Some applications for the SAT are non-traditional databases (e.g. storing images, fingerprints or audio clips, where the concept of exact search is not used and we search instead for similar objects); text searching (to find words and phrases in a text database allowing a small number of typographical or spelling errors); information retrieval (to look for documents that are similar to a given query or document); machine learning and classification (to classify a new element according to its closest representative); computational biology (to find a DNA or protein sequence in a database allowing some errors due to the mutations); and function prediction (to search for the most similar behavior of a function in the past so as to predict its probable future behavior).

A typical query for this data structure is the *range query* which consists on retrieving all objects within a certain distance from a given query object. The distance is expensive to compute and is usually the relevant performance metric to optimize, even over the secondary memory operation cost [11]. This problem is more significant in very large databases, making it relevant to study efficient ways of parallelization.

So, using data parallelism is another way to improve the multimedia web searches due to be one of the most successful efforts to introduce explicit parallelism to high level programming languages. The approach is taken because many useful computations can be framed in terms of a set of independent sub-computations, each strongly associated with an element of a large data structure. Such computations are inherently parallelizable. Data parallel programming is particularly convenient for two reasons. The first, is its easiness of programming. The second is that it can scale easily to large problem sizes.

In this work we have selected the SAT structure to index multimedia data, because the SAT is a nice example of tree data structure in which well-known tricks parallelization simply do not work. It is too sparse, unbalanced and its performance is too dependent on the workload generated by the queries being solved by means of searching the tree. The complexity measure is given by the number of distances computed to retrieve those objects close enough to the query. We propose an efficient parallel algorithm using the Bulk Synchronous Parallel-BSP [25] model to perform the parallel querying and data distribution. This model provides independence of the computer architecture and has been shown to be efficient in the application such as text databases and others [23]. The algorithms can be implemented using any modern communication library such as PVM, MPI or special purpose libraries such as BSPlib

or BSPpub.

In the next section we describe the metric space and the sequential SAT structure. In section 3 we describe the Parallel Model BSP and the server's architecture. The proposed strategy is explained in section 4. Experiments results are presented in section 5, the presented strategy is evaluated on an English and Spanish dictionary. The final section summarizes this work and suggests future work.

## 2 THEORETICAL CONCEPTS

In many cases similarity is modeled through the metric spaces and the object searches is performed through range queries or nearest neighbor queries.

Let $\mathbb{U}$ be a universe of *objects*, with a nonnegative *distance function* $d : \mathbb{U} \times \mathbb{U} \longrightarrow \mathbb{R}^+$ defined among them. This distance satisfies the three axioms that make $(\mathbb{U}, d)$ a *metric space*: strict positiveness ($d(x, y) = 0 \Leftrightarrow x = y$), symmetry ($d(x, y) = d(y, x)$) and triangle inequality ($d(x, z) \leq d(x, y) + d(y, z)$). The smaller the distance between two objects, the more "similar" they are. We handle a finite *dataset* $S \subseteq \mathbb{U}$, which is a subset of the universe of objects and can be preprocessed (to build an index). Later, given a new object from the universe (a *query* $q \in \mathbb{U}$), we must retrieve all similar elements found in the dataset. There are two typical queries of this kind:

*Range query:* Retrieve all elements within distance $r$ to $q$ in $S$. This is, $\{x \in S\ ,\ d(x, q) \leq r\}$.

*Nearest neighbor query (k-NN):* Retrieve the $k$ closest elements to $q$ in $S$. That is, a set $A \subseteq S$ such that $|A| = k$ and $\forall x \in A, y \in S - A, d(x, q) \leq d(y, q)$.

In this paper we are devoted to range queries. Nearest neighbor queries can be rewritten as range queries in an optimal way [16], so we can restrict our attention to range queries. The distance is considered expensive to compute (think, for instance, in comparing two fingerprints). Hence, it is customary to define the complexity of the search as the number of distance evaluations performed, disregarding other components such as CPU time for side computations, and even I/O time. Given a dataset of $|S| = n$ objects, queries can be trivially answered by performing $n$ distance evaluations. The goal is to structure the dataset such that we perform as few distance evaluations as possible.

A particular case of this problem arises when the space is a set of $D$-dimensional points and the distance belongs to the Minkowski $L_p$ family: $L_p = (\sum_{1 \leq i \leq D} |x_i - y_i|^p)^{1/p}$. For example $p = 2$ yields Euclidean distance. There are effective methods to search in those spaces [12, 5]. However, for roughly 20 dimensions or more those structures cease to work well. We focus in this paper on general metric spaces, although the solutions are well suited also for $D$-dimensional spaces. Moreover, regarding a $D$-dimensional space as a metric space reveals the true dimensionality of the dataset, which may be much lower than $D$, without the need of applying an expensive dimensionality reduction technique.

For general metric spaces, there exist a number of methods to preprocess the database in order to reduce the number of distance evaluations [9]. All those structures work on the basis of discarding elements using the triangle inequality, and most use the classical divide-and-conquer approach (which is not specific of metric space searching).

### 2.1 Sequential SAT

The SAT is a structure of this kind [3], based on another concept: rather than dividing the search space, approach the query spatially, that is, start at some point in the space and get closer and closer to the query. Apart from being algorithmically interesting by itself, it has been shown that the *sa-tree* gives better space-time tradeoffs than the other existing structures on metric spaces of high dimension or queries with low selectivity, which is the case in many applications.

The SAT construction starts by selecting at random an element $a$ from the database $S \subseteq U$. This element is set to be the root of the tree. Then a suitable set $N(a)$ of neighbours of $a$ is defined to be the children of $a$. The elements of $N(a)$ are the ones that are closer to $a$ than any other neighbour. The construction of $N(a)$ begins with the initial node $a$ and its *bag* holding all the rest of $S$. We first sort the bag by distance to $a$. Then we start adding nodes to $N(a)$ (which is initially empty). Each time we consider a new node $b$, we check whether it is closer to some element of $N(a)$ than to $a$ itself. If that is not the case, we add $b$ to $N(a)$. We now must decide in which neighbour's bag we put the rest of the nodes. We put each node not in $a \cup N(a)$, but in the bag of its closest element of $N(a)$. The process continues recursively with all elements in $N(a)$.

The resulting structure is a tree that can be searched for any $q \in S$ by spatial approximation for nearest neighbour queries. The mechanism consists in comparing $q$ against $a \cup N(a)$. If $a$ is closest to $q$, then $a$ is the answer, otherwise we continue the search by the subtree of the closest element to $q$ in $N(a)$.

Some comparations are saved at search time by storing at each node $a$ its covering radius $R(a)$, i.e., the maximum distance between $a$ and any element in the subtree rooted by $a$.

It is a little interest to search only for elements $q \in S$. The tree we have described can, however, be used as a device to solve range queries for any $q \in U$ with radius $r$. The key observation is that, even if $q \notin S$, the answer to the query are elements $q' \in S$. So we use the tree to pretend that we are searching an element $q' \in S$. Range queries $q$ with radius $r$ are processed as follows. We first determine the closest neighbour $c$ of $q$ among $\{a\} \cup N(a)$. We then enter into all neighbours $b \in N(a)$ such that $d(q,b) \leq d(q,c) + 2r$. This is because the virtual element $q'$ sought can differ from $q$ by at most $r$ at any distance evaluation, so it could have been inserted inside any of those $b$ nodes. In the process we report all the nodes $q'$ we found close enough to $q$. Finally, the covering radius $R(a)$ is used to further prune the search, by not entering into subtrees such that $d(q,a) > R(a) + r$, since they cannot contain useful elements.

Besides, we can use another improvement to prune the search. At any node $b$ of the search we keep the track of the minimum distance $d_{min}$ to $q$ seen up to now across this path, including neighbours. We enter only neighbours that are not farther than $d_{min} + 2r$ from $q$.

Figure 1 illustrates the search process, starting from $p_{11}$ (tree root). Only the element $p_9$ is in the result, but all the bold edges are traversed.

We depict below the algorithm to search an element $q \in U$ with radius $r$ in a SAT. It is firstly invoked as *RangeSearch* ($a$,$q$,$r$,$d(a,q)$), where $a$ is the root of the tree. It can be noticed that in the recursive invocations $d(a,q)$ is already computed.

```
1: procedure RANGESEARCH(Node a, Query q, Radius r, Dist. d_min)
2:     if (d(a,q) ≤ R(a)+r) then
3:         if (d(a,q)≤ r) then
```

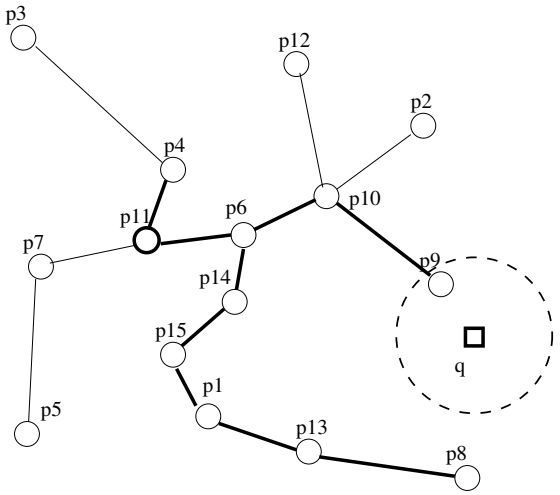



**Figure 1: An example of the search process in a SAT.**

```
 4:            report a
 5:        end if
 6:        d_min ← min{d(c,q),c ∈ N(a)}∪ d_min
 7:        for (b ∈ N(a)) do
 8:            if (d(b,q)≤ d_min+2r) then
 9:                RangeSearch(b,q,r,d_min)
10:            end if
11:        end for
12:    end if
13: end procedure
```

# 3 PARALLEL MODEL AND SERVER'S ARCHITECTURE

In the *Bulk Synchronous Parallel, BSP* model of computing, proposed in 1990 by Leslie Valiant [25], 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 [23].

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* [17] or *BSPpub* [18].

$BSP$ 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 $PVM$ [13] and $MPI$ [24] 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 [14, 15].

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.

Notice that the requirement of periodically barrier synchronizing the processors can be relaxed in situations in which a given processor knows beforehand the number of messages it should expect from all others. In this case, a given processor just waits until it receives the proper number of messages to further continue its computations on local data. Barrier of sub-sets of processors is also possible [25, 17].

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 [2, 26, 27].

We assume a server operating upon a set of $P$ machines, each containing its own memory. Client request services to a broker machine, which in turn distribute those request evenly onto the $P$ machines implementing the server. Request are queries that must be solved with the data stored on the $P$ machines. We assume that under a situation of heavy traffic the server start the processing of a batch of $Q$ queries in every superstep.

Basically every processor has to deal with two kind of messages, those from newly arriving queries coming from the broker, in which case a search is started in the processor, and those from queries located in others processors that decided to continue their search in a subtree of this processor.

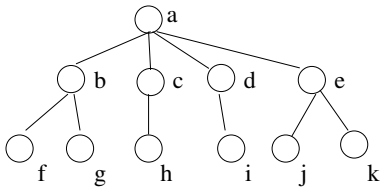
# 4 SEARCH STRATEGY

As we said before, multimedia web searches is an interesting issue but it has not been deeply researched, in fact there is very little work in this area. Therefore to develop a good engine search for any kind of object we must select a suitable data structure as an index, in this work we have selected the SAT structure. A first point to emphasize is that the SAT structure contains nodes of very diverse number of children. Every child node causes a distance comparison, so it is relevant to be able to balance the number of distance comparisons performed in every processor per superstep.

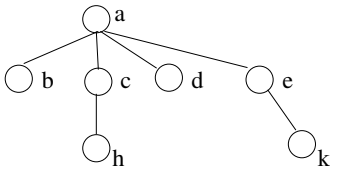
Thus in this work we present a strategy to map the tree nodes onto the processors by considering the number of distance comparisons that may be potentially performed in every subtree rooted at the children of the SAT's root. That is, the subtrees associated with nodes $b$ in $N(a)$ where $a$ is the root and $N(a)$ is the set neighbour of $a$.

To do that, we replicate the root and each child of the root in every processor and we distribute all the others nodes evenly through the processors, in a multiplexed way. A disadvantage is that every node has to replicate its children locally, to be able to perform the distance comparisons and in this way continue the searching operation (see Figure 2).

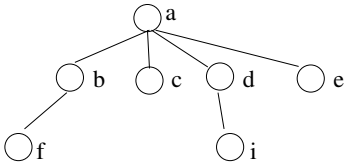
To improve efficiency we set an upper limit $V$ to the number of distance comparisons that are performed per processor in each superstep. During a superstep, every time any processor detects that it has performed more than $V$ distance comparisons, it suspends query processing



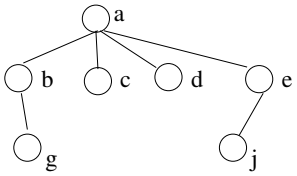

(a) Sequential SAT tree.




(b) Distributed SAT tree stored in the processor $P_0$.




(c) Distributed SAT tree stored in the processor$P_1$.




(d) Distributed SAT tree stored in the processor$P_2$.

Figure 2: Distribution of the nodes of the sequential SAT tree upon a server with three processors.

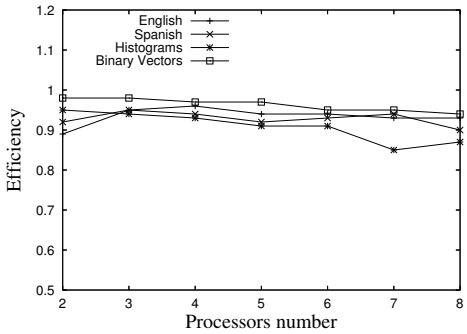
and waits until the next superstep to continue with this task. Under the BSP model it means that all queries going down in a tree in each processor $k$ has to be sent again to the processor $k$ as a message, exactly as if it found out that the search has to continue in other processor. But no communication cost is involved for these extra messages. Also the processor stop extracting messages from its input queue.

Besides, every $s$ supersteps we collect statistics that are used to define the value of $V$ for the next sequence of supersteps. This statistics are independent of the value of $V$ and of the $s$ supersteps used to calculate them. In this way the value of $V$ can adapt it self to the workload changes produced by the flow of queries arriving constantly to the server.

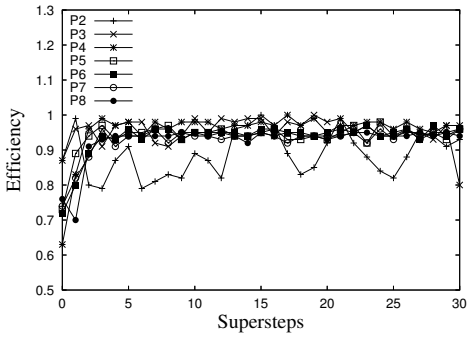
Because of the limits $V$, supersteps can be truncated before processing all the available queries. Therefore real supersteps are not a reliable measure of the real average number of supersteps required to complete a query.

To deal with this, we put in every query $q$ a counter of virtual supersteps different from the real ones executed by the BSP computer. Also, we keep counter for the virtual supersteps in each processor $k$. Every time a new query is initialized in a processor $k$ we set the virtual supersteps of the query to be equal to the number of batch it belongs to. The broker can do this before sending the query to the processor. Besides, every time a query has to migrate to another processor we increase the virtual supersteps in one unity, because it takes one virtual superstep to get there.

Additionally, we count the total number of distance calculations that has been performed in every processor $k$. It gives us a precise idea of global load balance (across supersteps).




(a) Efficiency archived by the multiplexed distribution with upper limits to the number of distance calculations with different databases.




(b) Efficiencies per supersteps. Proposed strategy of SAT distribution onto the processors and limits to the number of distance calculations per superstep with the English dictionary.

Figure 3: Efficiency archived by different size of server and per supersteps.

# 5 EXPERIMENTAL EVALUATION

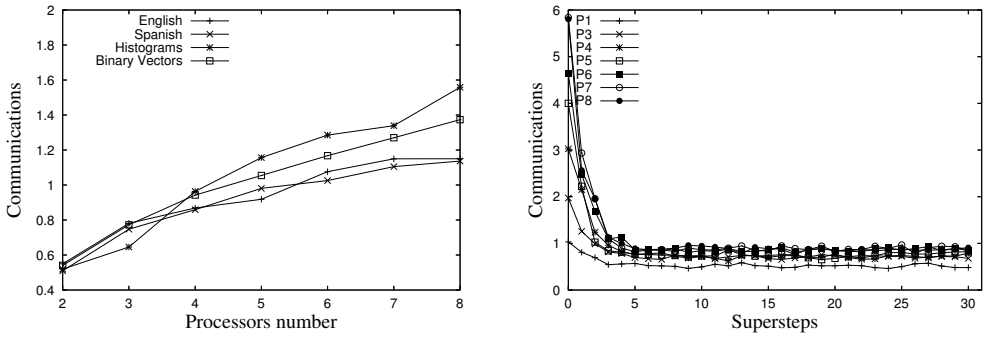
The load balance of the computations effected by the processors is the main measure used in our experiments. We define it as the efficiency $Ef$ and $Ef = 1$ indicates the optimal, as showed in equation 1

$$Ef = (\sum_{i=1}^{P} distances/P)/maximum\_number\_distances \quad (1)$$

where $P$ is the number of processors. The average of this measure is taken over all supersteps. We measure computation by considering the number of distance calculations among objects during query operations and communications is the number of message sent/received among processors.

In the experiments below, we use a 69K-words English dictionary and a 51K-words Spanish dictionary, where queries are composed by words selected at random. We also use a Spanish dictionary in the same way. In these cases the distance between two objects is the edit distance, that is, the minimum number of characters insertions, deletions, and replacements to make the two strings equal. Another databases used in the experiments is an 112682 histogram of images and binary vectors (100000 vectors on dimension 10) where the Euclidean distance is used to compute the distance between objects. We also assume a demanding case in which the broker circularly among the processors distribute queries. The SAT is initialized with the 90% of the database and the remaining 10% are left as query objects (randomly selected from the whole database).

Figure 3 shows results for the efficiency $Ef$ with the multiplexed distribution of the SAT's nodes with adaptive upper limit $V$. In Figure 3(a) we present the behavior of this distribution for different databases. Here the histogram database reports the lowest values as the server size is increased. In Figure 3(b) the efficiency archived per superstep is showed for different number of processors using the English dictionary. Here we can see the average number of distance calculations per supersteps in all processors. This measure tends to obtain a low value in the




(a) Communication archived by the multiplexed distribution with upper limits to the number of distance calculations with different databases.

(b) Messages sent/receives per supersteps. Proposed strategy of SAT distribution onto the processors and limits to the number of distance calculations per superstep using the English dictionary.

Figure 4: Communication archived by different size of server and per supersteps.

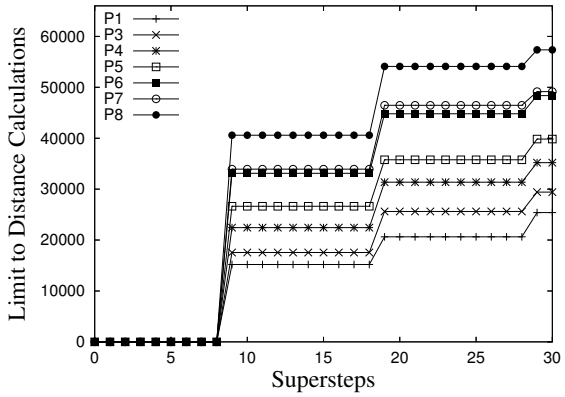



Figure 5: Automatic and adaptive calculation of limits V per supersteps.

firsts supersteps, but then it is close to the value one, the optimal. This indicates a good load balance of the computations.

Figure 4 shows the results for the amount of communications demanded by the multiplexed distribution. Remember that in this strategy nodes are placed in a circular manner independently of their father-child-brother relation. The communication is measure as the ratio $A/B$ where $A$ is the total number of messages sent between processors and $B$ is the total number of time a function $search()$ in charge of solving queries and calculating the distance, was called to complete the processing of all queries.

As expected, Figure 4(a) shows how communication grows as the number of processors is increased significantly. And in this case the histogram database is the one requiring more communication to solve the queries. In contrast, Figure 4(b) shows the communication reported by different number of processors per superstep. In this last case, the communication is high at the beginning but as the number of supersteps is increased, the communication tends to be small, close to one.

Figure 5 shows how the adaptive value of $V$ per supersteps. At the beginning all $V$ are set

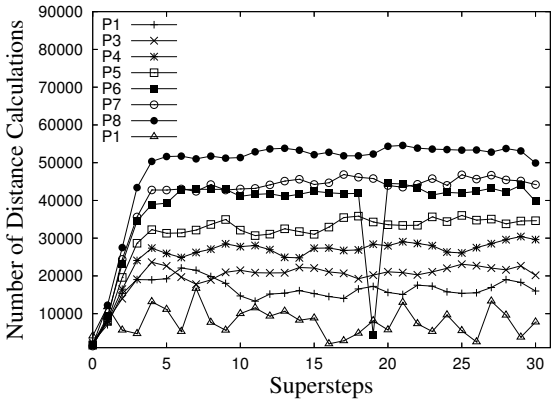



Figure 6: Maximum number of distance computations performed per supersteps.

to zero. Then these values self adapt to different values according to the distance calculations performed in each superstep. Every time a processor gets to this value, it has to stop the queries processing and wait for the next superstep to continue. As the number of processors is bigger the value $V$ is higher.

Finally, Figure 6 shows the maximum number of distance computations performed in each superstep by different size of the sever. Here the results obtained by the processor report a low value at the start of the experiment, but after the third superstep the processors begin to have many kind of queries to be sold. They receive queries from the broker, from other processors and they continue with the processing of unfinished queries truncated in previous supersteps.

# 6 FINAL COMMENTS AND FUTURE WORK

We have presented a distributed strategy to search for similar objects in a metric space. This strategy can be used in a web environment as an index structure for efficient search multimedia objects. This is an interesting area with very little research done so far.

We have selected the SAT structure to index this kind of elements and to improve the performance of the searches we have used the parallel paradigm through the BSP model. We have focused on balancing the number of distance calculations across supersteps because this is the most relevant performance metric to optimize. Distance calculations between complex objects are known to be expensive in running time. On the other hand, we have observed that the amount of communication and synchronization is indeed extremely small with respect to the cost of distance calculations.

The experiments were performed over three databases, the English, the Spanish dictionary and a histogram of images. The firsts two ones report a better efficiency and lower communication than the third one.

As future work we are going to research another parallelizations alternatives for the static SAT structure and also we are going to applied these method to the dynamic SAT.

# ACKNOWLEDGMENT

This work has been partially funded by research project CYTED red VII.J (RITOS2) and PICT2002-1260.

## REFERENCES


[1] Bharat K. amd Broder A. Mirror, mirror on the web: A study of host pairs wirh replicated content. In *In Proceedings of the Eighth World Wide Web Conference*, 1999.

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

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

[4] Berg A. Bar-Yossef Z. and Weitz J. F. D. Approximating aggregate queries about web pages via random walks. In *In Proceedings of the Twenty-sixth International Conference on -very Large Databases*, 2000.

[5] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, September 2001.

[6] Sergei Brin. Near neighbor search in large metric spaces. *The 21st VLDB Conference*, 1995.

[7] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communication of ACM*, 1973.

[8] S.Berchtold C. Bohm and D. Kein. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, (33(3):322-373), 2001.

[9] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[10] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *The 23st International Conference on VLDB*, 1997.

[11] G. Navarro D. Arroyuelo, F. Muñoz and N. Reyes. Memory-adaptative dynamic spatial approximation trees. *Proceedings of the 10th International Symposium on String Processing and Information Retrieval (SPIRE 2003)*, (LNCS 2857, pages 360-368), Springer, 2003.

[12] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[13] 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.

[14] 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.

[15] 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.

[16] G. R. Hjaltason and H. Samet. Incremental similarity search in multimedia databases. Technical Report CS-TR-4199, University of Maryland, Computer Science Department, 2000.

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

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

[19] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal (VLDBJ)*, (11(1):28-46), 2002.

[20] G. Navarro and N. Reyes. Improved deletions in dynamic spatial approximation trees. *In Proc. of the XXIII International Conference of the Chilean Computer Science Society (SCCC'09)*, (pages 13-22, IEEE CS Press), 2003.

[21] G. Navarro and N. Reyes. Fully dynamic spatial approximation trees. *In Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002), LNCS 2476*, (pages 254-270), Springer, 2002.

[22] Lawrence S. and Giles C. L. Accessibility of information in the web. In *Nature 400, 107-109*, 1999.

[23] 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.

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

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

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

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