# Empirical Evaluation of the Parallel EGnat Index

Veronica Gil-Costa Mauricio Marin
Yahoo! Research Latin America, Chile
Ricardo J. Barrientos
DCC, University of Chile, Chile
Carolina Bonacic
ArTeCS, Complutense University of Madrid, Spain

***Abstract*—Web sites such as flickr have to manage a great amount of multimedia data available. These applications work with specialized indexes capable of handling complex information. In this paper we evaluate a particular parallel multimedia index, the EGnat. We propose to extend the root node of this index and we compare the efficiency of the global and local parallel index distribution. We empirically show that the global approach archives better performance not only because how the index itself is distributed but also because global centers/pivots discard more non-similar objects to a query. We also test different techniques to select the centers and improve the EGnat performance.**

## I. Introduction

As the Web grows the information and data found in this repository are not only text, but images, sounds, videos and any kind of multimedia data. So, it is interesting to design a web search engine able to find any kind of objects, for example a user may introduce part of a song and recover the whole song. Or it could be more interesting and helpful for doctors trying to find part of a DNA, or similar X-rays. For the text retrieval problem there are many systems dedicated to recover useful information/documents for user queries as web search engines like Yahoo! or Google. In this context similarity searches are used to correct edition errors or try to find words not appearing in the dictionary, or words created by users.

Searching for similar objects may be performed over a metric space where objects are compared using the distance between them and a search radius $r$. An object is said to be similar to a query if the distance between them is less or equal than the search radius $r$. But due to the distance evaluation has a high computation cost, many data structures able to handle these unstructured objects have been presented in the literature [1], [2], [3], [5], [7], [13]. Each one of them has different features making suitable for specific applications (secondary memory access, static collections, and dynamic updates on the fly).

The distance between two database objects in a high-dimensional space can be very expensive to compute and in many cases it is certainly the relevant performance metric to optimize; even over the secondary memory operations cost. For large and complex databases it then becomes crucial to reduce the number of distance calculations in order to achieve reasonable running times. This makes a case for the use of parallelism.

There are basically two main strategies to parallelize indexes over a cluster of PCs. In the first one named *local* partition scheme, the whole collection is distributed among $P$ processors and then each processor builds its own index using the local data. At query processing time, the query has to be sent to all processors and similar objects are found using indexes built upon different objects. Notice, that this kind of strategy is not scalable because of the communication involved in the query search phase is $O(P)$ and each query use all the resources available in the system. Besides, sending a copy of the query to all processors increase the overhead involved with large number of threads and disk access. The advantage is that local indexes are easy to build and maintain since insertion of new objects can be done locally without any extra communication cost.

The *global* scheme is the other main technique used to distribute the data among processor. In this case, one sequential index is build upon the whole collection and then the index is distributed among processors. The way the index is distributed depends on its features. At a given instant any query is processed in a single machine and many queries may be processed in different machines. So we have a query level parallelism allowing to archive $O(1)$ scalability. However, index construction and maintenance is much more costly in communication.

The above discussion about local/global indexes is old in the literature of parallel computing, and in the context of Web engines using Inverted Files as the index we can find some contradictions like [14], [18]. This happens because the outcomes of the experiments are dependent of the particular implementation and platform used. This remarks the importance of using a parallel model to ensure fairness and independence from hardware and software realizations [12].

In this paper we extend a previous work [17] using the EGnat as the data structure index, evaluating the two main distributed techniques and applying a Web search engine architecture based in the round-robin principle. We compare different techniques to select the centers of the EGnat nodes and we empirically show that the global index discards more elements, because of its feature of selecting *special* objects using the whole database.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 presents the metric space and the sequential EGnat index with some improvements. Section 4 presents the parallel query organization and the two parallel approaches. Evaluation environment, test scenarios and results are presented in Section 5. Finally some conclusions and future work are drawn in Section 6.

## II. Previous Work and Motivation

Regarding previous work on parallel query processing on metric-spaces, a relevant paper is [19] where the problem is analyzed in the abstract. In [2] they present how to parallelize a pivot index. We have contributed with some work [9], [10], [11], [15]. These realizations were performed upon the Bulk Synchronous Parallel - BSP parallel model [20]. We have studied several forms of parallelization of the List of Cluster (LC) [10], the SAT [16] and the SSS [9] strategies. We have also presented some alternatives to build the global index in parallel in order to reduce the communication between processors and overcame its weakness.

In [9] we analyze secondary memory storage and a way to improve the pruning at search query time. In all these works the global partitioned scheme has presented better performance than the local scheme. The global approach has two main advantages. The first ones is that all centers/pivots of the index are selected using the whole collection, allowing to use more information during the pruning stage and avoiding enter zones without similar objects to the query. So, we say that global pivots/centers have a better overall quality and are better representatives of the collection. We present some experiments upon the EGnat and two others data structures to validate this claim. Another advantage of the global index, is that instead of having one query using all the machines resources (disk, CPU), we can process several queries at the same time.

In this paper we have selected the EGnat because the local partitioned scheme has been reported as a better parallel strategy than the global. But experiments in [17] were performed only with 10 processors, using small data sets and queries were selected from the same collection. Besides, each node size was $k = 20$, too small to prune subtrees especially in the root node. So, in this work we apply our search engine architecture in order to unify the algorithms and we run them over larger data sets, with more machines and using a real log of queries. Our hypothesis is that using a bigger root node we can select fewer processors to send the query and optimize the global index approach performance.

## III. Theoretical Concepts and Sequential Indexing

A *metric space* $(\mathbb{X}, d)$ is composed of an universe of valid objects $\mathbb{X}$ and a *distance function* $d : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}^{+}$ defined among them. The distance function determines the similarity between two given objects. The goal is, given a set of objects and a query, to retrieve all objects close enough to the query. This function holds several properties:

- Strictly positiveness: $d(x, y) > 0$ and if $d(x, y) = 0$ then $x = y$.
- Symmetry: $d(x, y) = d(y, x)$.
- Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$.

The finite subset $\mathbb{U} \subset \mathbb{X}$ with size $n = |\mathbb{U}|$, is called the database and represents the collection of objects. There are three main queries of interest

- *range search:* that retrieves all the objects $u \in \mathbb{U}$ within a radius $r$ of the query $q$, that is: $d(q, r) = \{u \in \mathbb{U} | d(q, u) \leq r\}$;

- *nearest neighbor search:* that retrieves the most similar object to the query $q$, that is $NN(q) = \{u \in \mathbb{U} | \forall v \in \mathbb{U}, d(q,u) \leq d(q,v)\}$;
- *k-nearest neighbors search:* a generalization of the nearest neighbor search, retrieving the set $k\text{-}NN(q) \subseteq \mathbb{U}$ such that $|k - NN(q)| = k$ and $\forall u \in k\text{-}NN(q), v \in \mathbb{U} - k\text{-}NN(q), d(q,u) \leq d(q,v)$.

We focus on range queries since nearest neighbor queries can be rewritten as range queries in an optimal way [6]. Well-known data structures for metric spaces can be classified as pivot or cluster based techniques. A pivot-based strategy selects some objects as $pivots$ from the collection and then computes the distance between the pivots and the objects of the database. This information is maintained in a table where each sell stores $d(i,j)$ for each pivots $i$ and object $j$. Several algorithms, like [2], [4], are almost direct implementations of this idea, differing in their extra structure used to reduce the CPU cost of finding the candidate points, but not in their number of distance evaluations.

Clustering techniques partition the collection of data into groups called $clusters$ such that similar entries fall into the same group. Thus, the space is divided into zones as compact as possible, usually in a recursive fashion, and this technique stores a representative point ("center") for each zone plus a few extra data that permit quickly discarding the zone at query time. Some clustering structures are the List of Clusters (LC) [5], the Geometric Near-neighbor Access Tree (GNAT) [1] and the MTree [8].

## A. EGnat

The Evolutionary Geometric Near-neighbor Access Tree (EGnat) index is an extension of the Gnat [1] devised to optimize secondary memory access, storage space and to reduce the number of distance evaluations. It can be seen as a tree with two kind of nodes: *buckets* or *bags* and *gnats*. To build the EGnat we first select $k$ objects from the data collection as centers or splits in order to divide the space in compact zones. All nodes are initially created as buckets maintaining only the distance to their fathers. When a bucket becomes full it evolves from a bucket node to a gnat one by re-inserting all its objects into the new gnat node.

Then for each object in the collection we select the closest center $c_i$ to the object and we put it in the bag of the center. We denote $D_{c_i}$ the bag for a center $c_i$. For each pair of centers $(c_i, c_j)$ we obtain a range table $range(c_i, D_{c_j}) = [min\_d(c_i, D_{c_j}), max\_d(c_i, D_{c_j})]$ where $min\_d$ and $max\_d$ are the minimum and maximum distance $d(c_i, x)$ between the center $c_i$ and any object $x \in D_{c_j} \cup \{c_j\}$. Every set $D_{c_j}$ is a sub-tree with root $c_j$ and the EGnat tree is built recursively in each sub-tree $D_{c_j}$.

Searches are performed as follow. For a query $q$ with radius $r$, we want to recover all objects $o \in S$ at a distance $d \leq r$. If the search is performed over a bag node we can use the distance evaluation to discard objects not close enough to the query and avoid computing the distance evaluation between the query and the buckets objects. So, we can discard an object $s_i$ belonging to a sub-tree with root $c_i$ if $d(s_i, c_i) > d(q, c_i) + r$ or $d(s_i, c_i) < d(q, c_i) - r$. If this condition does not hold we have to perform the distance evaluation between the query $q$ and the object. If the node is of type *g*nat, the search is performed recursively as shown in Algorithm 1.

**Algorithm 1** *EGnat*: Searching.

```
rangesearch(Node P, Query q, Range r)
  {R : Set of results.}
  $R \leftarrow \emptyset$
  $d \leftarrow d(p_0, q)$
  if $d <= r$ then
    report $p_0$
  end if
  $range(p_0, q) \leftarrow [d - r, d + r]$
  for all $x \in P$ do
    if $range(p_0, q) \cap range(p_0, D_{p_x}) \neq \emptyset$ then
      add $x$ to $R$
      if $d(x, q) <= r$ then
        report $x$
      end if
    end if
  end for
  for all $p_i \in R$ do
    rangesearch($D_{p_i}$,q,r)
  end for
```

## B. Sequential Improvement

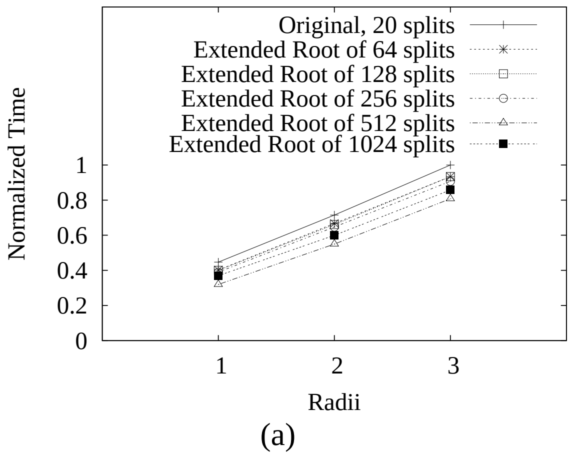
We introduced an improved root at the EGnat presented in [17]. First we extended number of splits in the root, and as is shown in the Figure 1(a) this long size allows a better performance. For the experiment presented in Figure 1(a) we used a collection of $955,705$ English words using the edit distance as the distance function with radii 1,2 and 3, and we processed $10,000$ queries selected from a sample of the UK Web. In Figure 1(b) we used the NASA collection with $1,000,000$ objects applying the Euclidean distance with radii 0.47, 0.57 and 0.73 in order to retrieval 0.01%, 0.1% and 1% of the data set objects. With the NASA collection we built the index with 90% of objects selected at random and remaining were used as queries. Of course there is a maximum limit over the root size, because if we extend it to much we would perform almost a sequential search. The best root size for both collections is $k = 512$.

There is a trade-off between the search performance and the index size. With an extended root we discard a greater number of sub-trees at the root level, but we require more memory space due to the range table maintained in each node of the EGnat. Notice that the range table of a node is a matrix of size $k^2$ ($k$= number of splits).

Figure 2 shows different techniques to select the centers of the root in the EGnat using the word collection and three search radius. We test several algorithms like $Max$ where we select an object as a center if it maximize the sum of distance to previous centers; our proposal $SSS - MaxMin$ selects the centers using the SSS rule and associates the rest of the objects to each center. Then if the number of elements associated to a center is less than $Min$ we remove the center and we select others centers for the elements. If the number of elements associated to a center is greater than $Max$ we adjust the parameter $M$ of the SSS technique. Finally, we compare it with the SSS [2] technique, which performs less number of distance evaluations. We obtained the same tendence for the NASA collection.

## IV. Query Processing Architecture

In this paper we use an efficient form of parallel computing to evaluate our query processing technique. Our comparative study is effected on top of the bulk-synchronous model of parallel computing (BSP) [20]. Its main advantage stems from the fact that BSP provides a cost model that allows a seemingly precise evaluation of the communication and computation costs of parallel programs, where 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.



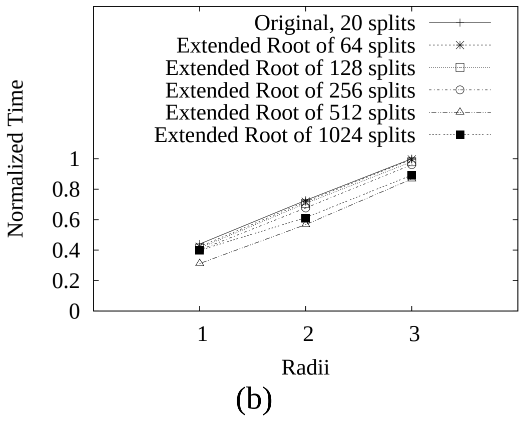


Fig. 1. (a)Running time extending the root size using the English collection and (b) using the NASA collection.

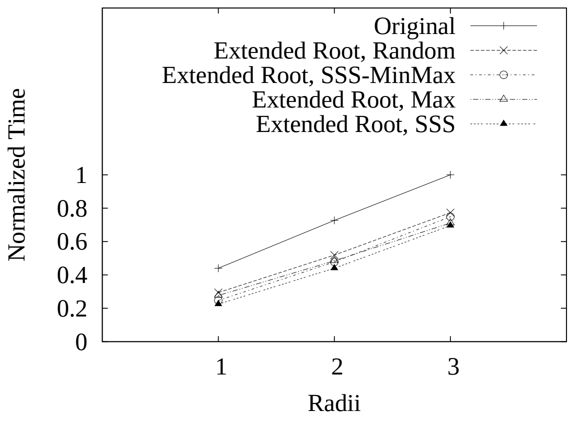


Fig. 2. Running time with extended root using different centers selection techniques.

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.

The total running time cost of a BSP program is the cumulative sum of the costs of its supersteps, and the cost of each superstep is the sum of three quantities: $w$, $hG$ and $L$, where $w$ is the maximum of the computations performed by each processor, $h$ is the maximum of the 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 $P$, as the communication time increases at least $O(log(P))$. In a situation of high traffic, we can assume that the broker distribute $Q = q{\cdot}P$ queries in every superstep so that $q$ new queries arrive at each processor in each superstep. In this case, it is not difficult to see that the cost of the broadcast operation we employ is $O(qP + qPG + L)$ against the $O(qP + qP2G + L)$ common practice strategy reported in the literature.

BSP allows us to apply the round-robin principle to search operations. We refer to the classic round-robin strategy for dealing with a set of jobs competing to receive service from a CPU. In this case each job is given the same quantum of CPU so that jobs requiring large amounts of processing cannot monopolize the use of the CPU. This scheme can be seen as bulk-synchronous in the sense that jobs are allowed to perform a fixed set of operations during their quantum. This prevent for long queries to monopolize the system resources for a long time.

In our setting we define quanta in computation, disk accesses and communication for the search operations which enables a better utilization of resources while it improves response times for queries. This requires careful consideration of the most costly parts of the solution to queries in very large distributed metric-space databases which are secondary memory management and load balance of distance calculations across processors.

We assume a server operating upon a set of $P$ machines, each containing its own memory. Client request are sent to a broker machine, which in turn distribute those request evenly onto the $P$ machines implementing the server. Requests are queries that must be solved with the data stored on the $P$ machines. The broker assigns a processor to be the *ranker* for each query in a circular manner. Each processor has to fetch the nodes of the EGnat, and has to perform the distance evaluation to determine similar objects to a query.

### *A. Local Index Approach*

The data set is distributed among $P$ processors and then each processors builds its own local EGnat index. When a new query arrives to the system the broker sends it to the ranker processor. Once this processor receives the query it is broadcasted to all processors. This first superstep has a cost $q{+}qG{+}L$. From the second superstep on, all processors begin a sequence of iteration, to obtain the *candidate* objects to the query and finally determine the similarity between them. Candidate objects are the ones that can not be discarded by the triangle inequality, so they have to be compared against the query.

Only one node of the EGnat is visited for any query per iteration. So the node is the quantum assigned to each query in order to apply the round-robin principle. Assuming an average number of $S_q$ sub-trees visited by any query $q$ and an average number of candidate centers $C_q$ per query and node, the cost of these supersteps is $[qk + qC_q + qC_qG + L] \cdot S_q$. Where $k$ is the maximum number of centers in any EGnat node, so we have to apply the triangle inequality to $k$ centers in order to select the candidate ones. Once we have determined the candidate centers $C_q$ in a node, we have to compare the query against them and if $d(q, c_i) < r$ then the center is part of the result. Then we have to continue the search in the sub-tree of the center $c_i$.

When a processor finds similar objects for a query, it sends them to the ranker machine. So we have to add the cost of sending $R_q$ results to the ranker machine. The asymptotic cost for a single query is:

$$Cost_{Local} = [k{+}C_q]{\cdot}S_q{+}[(C_q{+}R_q){\cdot}S_q{+}P]{\cdot}G{+}L{\cdot}S_q \tag{1}$$

### *B. Global Index Approach*

In the global index approach first we have to build the sequential index and then we multiplex the sub-trees of the EGnat root among $P$ processors. The first node of the EGnat (the root) is replicated in all processors. To maintain the necessary information required to jump from the root of a sub-tree to the node storing that sub-tree we maintain a hash table with the memory position of each node.

When a new query arrives to the system, the broker sends it to the ranker machine. Then the ranker determines the candidate centers and the processors holding them. Then the ranker sends a message to each processor holding a candidate center with the query, indicating where it has to continue. So the cost of this superstep is $q + qk + qC_q + C_qG + L$.

Notice that after the first superstep, many processors may continue with a query processing.

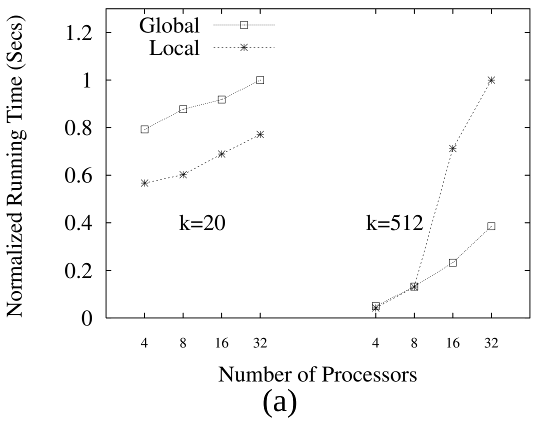
$$Cost_{Global} = [k+C_q]\cdot S_q + [(C_q+R_q)\cdot S_q]\cdot G + L\cdot S_q \quad (2)$$

Therefore, it is expected that the global approach presents better performance, because it can avoid the broadcast cost included in the local scheme. Another feature of the global index not reflected in these equations is the possibility of parallelize the use of the resources. All this depends on the pruning that takes place at the root node avoiding visiting all processors for a single query.

## V. Experiments

We used a collection of $955,705$ English words, using the edit distance as the distance function with a radius $r = 1$ and a collection of $1,000,000$ images taken from the NASA image and video archives. Each image is represented by a feature vector of $20$ components obtained from the colour histogram of the image. This index was built with 90% of the objects taken at random and the remaining were used as queries. The Euclidean distance was used with this collection to measure the similarity between two images with a radius $r = 0.47$ in order to retrieval in average 0.01% of the data set objects for each query. With the English collection we use a log with $10,000$ queries taken from the UK Web by Yahoo! search engine. The experiments were performed in a cluster with 32 processors Intel(R) Xeon(TM) CPU 2.80GHz with 512 KB cache size and 4Gb of RAM. Every processor complete $q = 10,000$ queries, so the whole number of queries processed by the system is $q{\cdot}P$, so it is expected that the total running time grows up with the number of processors.

Figure 3(a) shows the running time using $k = 20$ splits in the root and $k = 512$ splits at the right



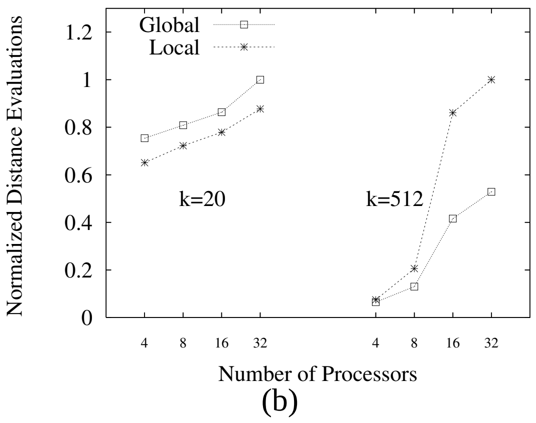
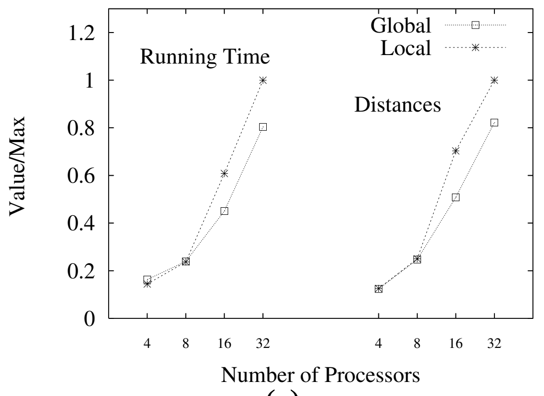


Fig. 3. Normalized running time and number of distance evaluations for the EGnat index using a root node with $k = 20$ splits and $k = 512$ splits.

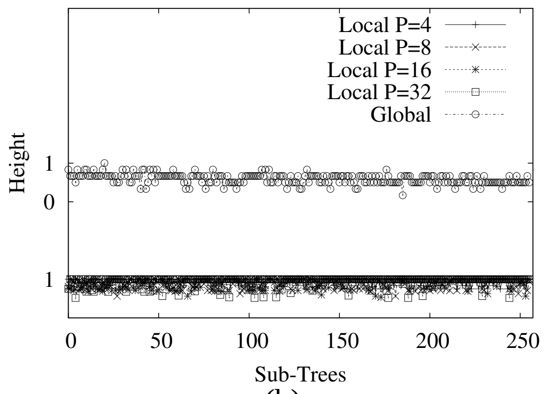
using the cluster with 4,8,16 and 32 processors. This graphic shows our hypothesis about the importance of the root node size in the global index. We can see that using $k = 512$ the running time is reduced for global strategy (when $P$ is great enough), mainly because the good selection of centers let a major discard of sub-trees, and this implies less processors are visited with each query.

The key to optimize the performance of the global index, besides the use of the resources in parallel, is based on the selection of the centers using the whole collection, as confirmed by Figure 5. Figure 3(b) shows number of distance evaluations for the same experiment.

Figure 4(a) shows the running time and the number of distance evaluations for the NASA collection using $k = 512$ splits in the root node. Figure 4(b) shows for each center of the root node $(sum(h_i/max_h))/P$, where $h_i$ is the sub-tree height in processor $i$, $max_h$ is the maximum height of



(a)



(b)

Fig. 4. (a)Running time and distance evaluations using the NASA collection; (b) Height of the root node sub-trees.

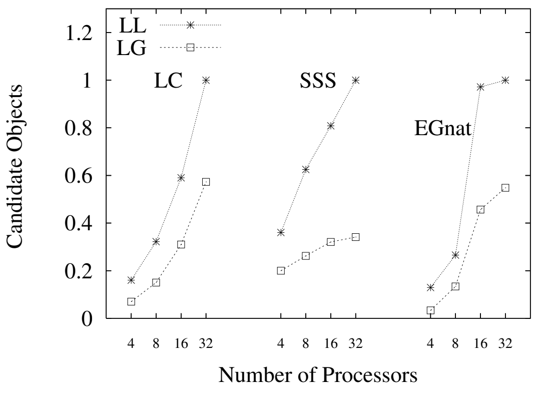


Fig. 5. Candidate objects for the local/global schemes.

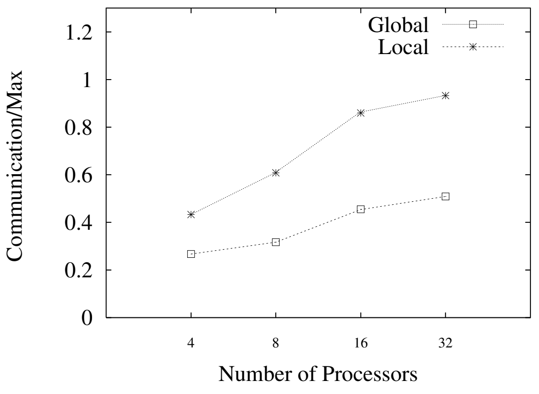


Fig. 6. Communication consumed by the local and Global index strategies during the query process.

the sub-tree in all processors and $P$ is the number of processors. This value indicates how different heights of sub-trees in all processors are, a value close to one indicate that heights are similar. For the global index, this graphics shows $h_i/max_h$ due to there is only one index distributed among processors.

## A. Quality of centers

To compare the quality of centers/pivots selected using a local or global approach we performed the next experiment. First we build the index using distributed local data for the local approach and we compare the pruning effect using the same distribution but with global centers/pivots (selected using the whole collection).

Figure 5 shows the average number of elements not discarded by the triangle inequality using a local index local centers ($LL$) and local index global centers ($LG$) approaches with the parallel LC [10], parallel SSS [9] and the parallel EGnat using $k = 512$ splits at the root node. For the LC we show the number of candidate clusters selected for a query using both approaches (number of zones of the metric space visited by the query). For the SSS we show the number of candidate objects for the query (objects that must be compared against the query), and for the EGnat we show the number of candidate centers (splits) selected with a root node size $k = 512$. For the three structures we can see that using global information allows discarding more elements. Note that for the pivot structure the difference is greater because in this kind of index we store the distance between the object and the query, but in clustering techniques we can only use the covering radius and objects inside the clusters must be compared with the query.

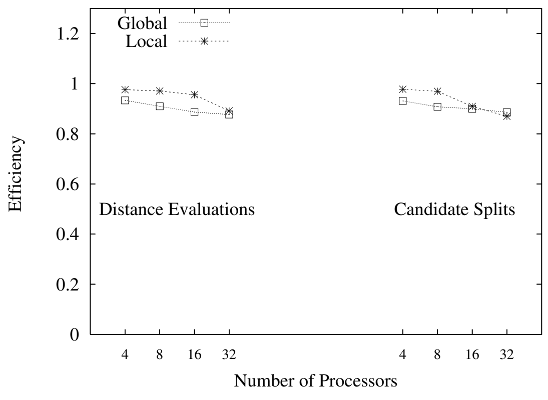


Fig. 7. Efficiency archived by the local and global approaches.

## B. Additional Metrics

Figure 6 shows the communication in bytes transmitted by the local and global index approaches during the query processing time. As it is expected the amount of bytes communicated grows up with the number of processors. The local approach presents more communication. Finally, Figure 7 shows the efficiency archived by both strategies using as efficiency measure metric the number of distance evaluation (at left) and the number of candidate splits processed per node (at right). In both cases, the local strategy tends to present a better efficiency due to all processors process the same queries.

## VI. Conclusions

We have applied our search engine architecture to the EGnat index, to unify the algorithms and compare the local and global partitioned scheme in a fair way. We have proposed an improved EGnat and also we have empirically show that that performance of a parallel search algorithm does not depend only in the data collection size or the number of processors, but in the root node size for the EGnat index case. The global scheme advantage not only lies in the type of space partitioning, but also because the centers/pivots are selected using the whole collection.

We are currently evaluating the gain in performance in this architecture by solving queries using the standard openMP. However our improvement has a weakness because the size of the root node requires more memory and in secondary memory storage will require seeking more than one disk block. Also, we are intended to improve the load balance of the global approach using scheduling algorithms.

## References

[1] S. Brin. Near neighbor search in large metric spaces. In *21st conference on Very Large Databases*, 1995.

[2] N. R. Brisaboa, A. Farina, O. Pedreira, and N. Reyes. Similarity search using sparse pivots for efficient multimedia information retrieval. In *ISM '06: Proceedings of the Eighth IEEE International Symposium on Multimedia*, pages 881–888, Washington, DC, USA, 2006. IEEE Computer Society.

[3] B. Bustos, G. Navarro, and E. Chavez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003. Elsevier.

[4] E. Chavez, J. Marroquin, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In *Proc. 6th International Symposium on String Processing and Information Retrieval (SPIRE'99)*, pages 38–46. IEEE CS Press, 1999.

[5] E. Chavez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

[6] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquyn. Searching in metric spaces. *ACM Computing Surveys*, 3(33):273–321, 2001.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435, 1997.

[8] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435, Athens, Greece, August 1997. Morgan Kaufmann Publishers, Inc.

[9] V. Gil-Costa and M. Marin. Distributed sparse spatial selection indexes. *In 16th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2008)*, IEEE-CS Press:440–444, Toulouse, France, Feb. 13-15, 2008.

[10] V. Gil-Costa, M. Marin, and N. Reyes. Parallel query processing on distributed clustering indexes. *Journal of Discrete Algorithms (Elsevier)*, 7(1):3–17, 2009.

[11] V. Gil-Costa, M. Marin, and N. Reyes. An empirical evaluation of a distributed clustering-based index for metric space databases. *In International Workshop on Similarity Search and Applications (SISAP 2008), IEEE-CS Press*, pages 386–393, Cancun, Mexico, April 11-12, 2008.

[12] S. Hazelhurst. Reporting performance of computer programs: algorithms and the impact of architecture. *South African Computer Journal(To Appear)*, 2008.

[13] I. Kalantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, 9:631–634, 1983.

[14] M. Marin and V. Gil-Costa. High-performance distributed inverted files. *In Proc. 16th Conference on Information and Knowledge Management (CIKM 2007). ACM*, page 935938, 2007.

[15] M. Marin, V. Gil-Costa, and C. Bonacic. A search engine index for multimedia content. *In Proc. 14th European Conference on Parallel and Distributed Computing (EuroPar 2008)*, LNCS 5168:866–875, Aug. 2008.

[16] M. Marin and Nora Reyes. Efficient parallelization of spatial approximation trees. In *International Conference on Computational Science (1)*, pages 1003–1010, 2005.

[17] M. Marin, R. Uribe, and R. J. Barrientos. Searching and updating metric space databases using the parallel egnat. In *International Conference on Computational Science (1)*, pages 229–236, 2007.

[18] A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 348–355, 2006.

[19] A. Papadopoulos and Y. Manolopoulos. Distributed processing of similarity queries. *Distributed and Parallel Databases*, 9(1):67–92, 2001.

[20] L.G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33:103–111, Aug. 1990.