Fast Parallel k-NN Search in High-Dimensional Spaces

Hyun-Hwa Choi, Seung-Jo Bae
High Performance Computing Software Team
Electronics and Telecommunications Research Institute
Daejeon, Rep. of Korea
{hyunwha, sbae} @etri.re.kr

Kyu-Chul Lee
Dept. of Computer Science
Chungnam National University
Daejeon, Rep. of Korea
kclee@cnu.ac.kr

***Abstract*—We are currently witnessing a rapid growth of image data, triggered by the popularity of the Internet and the huge amount of user-generated content from Web 2.0 applications. To address the demanding search needs caused by large-scale image collections, two major approaches for high-dimensional data in cluster systems have been proposed: Speeding up the search by using distributed index structures, and speeding up the search by scanning a Vector Approximation-file (VA-file) in parallel. We propose to combine both techniques to search for large k-nearest neighbors (k-NN) in a high-dimensional space. We develop a distributed index structure, called a Distributed Vector Approximation-tree (DVA-tree), with a two-level structure: the first level is a hybrid spill-tree consisting of minimum bounding spheres, the second level is VA-files. We also introduce a new approximate k-NN search algorithm on this structure and derive cost formulae for predicting the response time of the k-NN search. We then provide a detailed evaluation on large, high dimensional datasets. In an experimental evaluation, we show that our indexing scheme can handle approximate k-NN queries more efficiently for high-dimensional datasets.**

Keywords-knn search; distributed indexing structure; high dimensionality

I. INTRODUCTION

We are currently witnessing a rapid growth of image data, triggered by the popularity of the Internet and the huge amount of user-generated content from Web 2.0 applications. Given such image collections, performing similarity search to find objects most similar to a given object is a classical problem with many practical applications. A common approach to similarity search is to extract so-called features from the objects, e.g., color, shape and texture information, and to utilize special index structures for these features.

To address the demanding search needs caused by large-scale image collections, several distributed index structures for high-dimensional data spaces have been proposed. Most of the approaches recently published focus mainly on supporting range queries or operating in peer-to-peer systems [1 - 4]. However, in order to provide similarity on massive high-dimensional data in cloud computing services or web search services, we need efficient ways of providing a k-nearest neighbors (k-NN) search for high-dimensional data in cluster environments. The k-NN search is a central requirement in database applications such content-based multimedia retrieval, because it has no input parameters that require prior knowledge of data. The "best" indexes have the following properties:

- The index should be deployable over multiple nodes in cluster environments.
- The index should require no special tuning of parameters required for each specific dataset.
- The set of candidates retrieved by the index should contain the most similar objects to the query.
- The number of candidates retrieved must be as small as possible, to reduce I/O and computation costs.

Over the years, little work for providing an efficient and scalable access to high-dimensional data in centralized systems have been done on the parallelization of trees or Vector Approximation-files (VA-files). In [5 - 6], the authors used R-trees [7] as underlying data structure, because they guarantee good space utilization and treat geometric objects as a whole. Koudas et al. [5] proposed a "Master R-tree" architecture. A master server contains all the internal nodes of the parallel R-tree, and the leaf level nodes are declustered across several data servers. The major focus of the work is on finding the optimal declustering "chunk size". Schnitzer et al. [6] designed a "Master Client R-tree" as parallel multi-dimensional indexing structure. The Master Client R-tree is a two-level distributed R-tree that has a single global index on a master server and local indexes on the other data servers. The Master Client R-tree is similar to the Master R-tree in the sense that it declusters leaf level nodes across data servers. However each data server creates a complete R-tree as its own local index using the leaf level nodes that are assigned to it. Liu et al. [8] introduced a parallel version of a hybrid spill tree. A top-tree is built on the sample feature vectors. Each leaf node in this top-tree then defines the partition, for which a hybrid spill-tree is built on a separate machine.

On the other hand, most multi-dimensional indexing structures have an exponential dependence upon the number of dimensions. In recognition of this, a VA-file [9] was developed to accelerate the scan through the feature vectors. The VA-file consists of two separated files: the vector file containing the feature vectors, and the approximation file containing a compressed representation of each feature vector. Nearest neighbor queries are processed using two phases. In the filtering phase, the entire approximation file is scanned sequentially to prune the majority of feature vectors. The candidates that cannot be pruned are refined by a random search of the vector file in the refinement phase. Weber et al. [10] and Chang et al. [11] proposed a parallel NN-search based on the VA-file to achieve a linear increase on search speed as the number of servers grows. However, the query response times of these solutions have not been satisfactory for a search engine which enables similarity search on the World-Wilde Web.

In this paper, we present a new distributed indexing structure for fast nearest neighbor search in high-dimensional feature space, called a Distributed Vector Approximation-tree (DVA-tree). The core problem of designing a fast parallel nearest neighbor search algorithm is to find an adequate clustering algorithm which distributes the data onto the nodes such that the data, which have to be read in executing a query, are distributed as equally as possible among the nodes. We create a sample small enough to fit on a single machine from large-scale feature vectors and build a hybrid spill tree on the sample. The feature vectors partitioned to each cluster by the built hybrid spill tree are stored into a separate machine. A local index server, which operates on the separate machine, manages a VA-file as local index to process k-NN queries. We also describe how parallel k-NN search based on the DVA-tree works and derive cost formulae for predicting the response time of the parallel k-NN search. We present an experimental evaluation of our indexing scheme using both real and synthetic data sets, and compare it against previous techniques. The experimental results show that our indexing scheme can handle approximate k-NN queries more efficiently for high-dimensional datasets.

The remainder of this paper is organized as follows. In the next section, we first define the similarity queries and briefly present existing methods for similarity query processing. In Section III, we introduce our newly proposed DVA-tree structure. We also present the approximate k-NN search operation on the DVA-tree and derive cost formulae for predicting the response time of the k-NN search. Section IV reports the findings of an experimental study conducted to evaluate the proposed scheme. Finally, in Section V, we draw some conclusions.

II. PRELIMINARY

A promising and widely used approach for similarity searching in multimedia databases is to map the multimedia objects into points in a metric space. The metric spaces include high-dimensional vector spaces, where objects are compared using Euclidean (L_2) distance.

A metric space *M=(D, d)*, where *D* is a domain of objects and *d* is a total distance function with the following properties:

Symmetry: $d(O_x, O_y) = d(O_y, O_x)$

Non negativity: $d(O_x, O_y) > 0$ ($O_x \neq O_y$) and $d(O_x, O_x)=0$

Triangle inequality: $d(O_x, O_y) \leq d(O_x, O_z) + d(O_z, O_y)$

The distance between two points *P* and *Q* in the metric space is defined by Euclidean distance function:

$$L_2(P,Q) = d(P,Q) = \sqrt[2]{\sum_{i=0}^{d-1}(Q_i - P_i)^2} \quad (1)$$

The similarity queries in a *D*-dimensional space may be defined as follows:

Definition II.1 Range Query

Given a query object $Q \in D$ and a maximum search distance *r*, the range query ***range(Q, r)*** selects all indexed objects O_j such that $d(O_j, Q) \leq r$.

Definition II.2 k- Nearest Neighbors (k-NN)

Given a query object $Q \in D$ and an integer $k \geq 1$, the k-NN query ***NN(Q, k)*** selects the *k* indexed objects which have the shortest distance from *Q*.

In order to achieve a better performance of k-NN search, two classes of techniques (index structures and scan methods) have been proposed for high-dimensional data. The basic idea of most high-dimensional indexing structures is to construct a tree structure by partitioning the data space or clustering data. These methods can prune the search space for queries using the partitioning. In a M-tree [12], for each routing object O_r, there is an associated sub-tree $T(O_r)$, called the *covering tree* of O_r. All objects in the covering tree $T(O_r)$ are within the distance *r* from O_r, r > 0. Given a query *Q*, a lower bound $d_{min}(T(O_r))$ on the distance of any object in $T(O_r)$ from *Q* is

$$d_{\min}(T(O_r)) = \max\{d(O_r, Q) - r, 0\} \quad (2)$$

Upper bound is

$$d_{\max}(T(O_r)) = d(O_r, Q) + r \quad (3)$$

Consider the largest distance d_k in current nearest neighbors. At the execution of k-NN search, the order in which nodes are visited can be determined by selecting the node for which the d_{min} lower bound is minimum, and any sub-tree for which $d_{min}(T(O_r)) > d_k$ can be pruned from the search. According to experimental observations, these lead to better performance.

The scan based VA-File [9] divides the data space into 2^b rectangular cells where *b* denotes a user specified number of bits. For each dimension *j*, a small number of bits (b_j) is assigned. There are 2^{bj} partitions along dimension *j*, requiring $2^{bj} + 1$ marks, i.e., $m_j[0]$, …, $m_j[2^{bj}]$. An approximation *a* for a data point *p* is generated as follows. Let a_j be the number of the partition into which p_j falls. A point falls into a partition only if it lies between the lower and upper bounds of that partition:

$$m_j[a_j] \le P_j < m_j[a_j + 1] \tag{4}$$

The approximation *a* is simply the concatenation of the binary b_j-bit patterns for each partition.

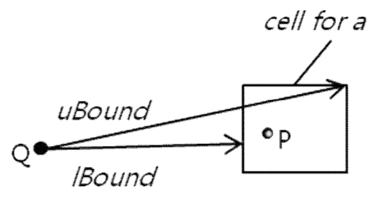

Figure 1. lower and upper bounds for L_2(Q, P)

In Fig.1, the lower and upper bounds on the distance between a query point *Q* and a data point *P* are determined by the equations:

$$lBound \le L_2(Q,P) \le uBound \tag{5}$$

$$lBound = \sqrt[2]{\sum_{j=1}^{D} (lBound_j)^2}$$
$$\text{where} \quad lBound_j = \begin{cases} Q_j - m_j[a_j + 1] & a_j < Q_j \\ 0 & a_j = Q_j \\ m_j[a_j] - Q_j & a_j > Q_j \end{cases} \tag{6}$$

$$uBound = \sqrt[2]{\sum_{j=1}^{D} (uBound_j)^2}$$
$$\text{where } uBound_j = \begin{cases} Q_j - m_j[a_j] & a_j < Q_j \\ \max(Q_j - m_j[a_j], m_j[a_j + 1] - Q_j) & a_j = Q_j \\ m_j[a_j + 1] - Q_j & a_j > Q_j \end{cases} \tag{7}$$

In the VA-file, the approximations are scanned linearly. A feature vector is a candidate whenever less than *k* feature vectors have been encountered, or whenever the lower bound is less than the k-th largest upper bound currently in the candidate set. The actual distance based on L_2 is evaluated only for these candidate feature vectors. In practical experiments, between 95% and 99% of the feature vectors were eliminated during scan step of approximations. The main advantage of the VA-file is that it retains good performance as dimensionality increases.

III. THE DVA-TREE

In order to improve the data access performance through the benefit of parallel process, it is important to distribute large data across multiple machines. For skewed distributions, the data density in some parts of a data space is higher than in other parts. Therefore, the core idea for a distributed indexing structure is to find an adequate clustering algorithm which distributes the data onto the nodes such that the data, which have to be read in executing a query, are distributed as equally as possible among the nodes. On the other hand, a sequential scan is superior to tree-based structures on a single machine, if the dimensionality of feature vectors exceeds a certain threshold [9]. We employ a tree structure as a clustering strategy. The tree is utilized for query processing, as usual, to restrict the search to relevant parts of the data space. The data points in each leaf node of the tree are stored into a separate machine with the VA-file.

A. The Strucuture

The structure of a DVA-tree is illustrated in Fig. 2. The DVA-tree is a distributed version of a two-level index scheme. The first level is a hybrid spill-tree consisting of minimum bounding spheres, the second level contains data points in a compressed representation.

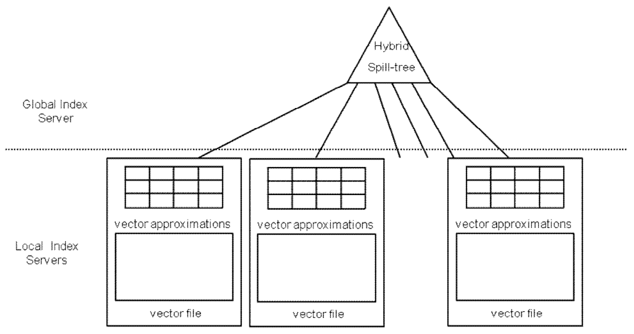

Figure 2. Structure of the DVA-tree

We first create a sample small enough to fit on a single machine from large-scale feature vectors. To accurately predict clusters of the entire feature vectors, we use the subset obtained from the feature vectors using random sampling method, and then build a hybrid spill tree on the sample. The hybrid spill-tree is the latest data partition method that is efficient in both accuracy and time of retrieval. The feature vectors partitioned to each cluster by the built hybrid spill tree are stored into a separate machine. Each of the separate machines manages a VA-file as local index to process a k-NN queries. The overall DVA-tree can be viewed conceptually as a single hybrid spill-tree, spanning a large number of machines.

B. K-NN Queries

The k-NN queries are processed by the three phases as shown in Fig. 3. In the first phase, the k-NN queries are submitted to the global index server owning the hybrid spill-tree. The global index server traverses the hybrid spill-tree in order to determine which VA-file(s) must be accessed. At this time, the global index server transforms the k-NN queries into range queries with arbitrary thresholds. The thresholds for the range queries are the average k-th distance between the sample data. They are computed while building the hybrid spill-tree. In the DVA-tree, whole clusters can be pruned by traversing the hybrid spill-tree. The k-NN queries are forwarded to the local index servers determined by the global index server. In the second phase, the local index servers process the k-NN queries on the VA-files in parallel. In the third phase, final results of the k-NN queries are obtained from candidate neighbors returned by the multiple local index servers.

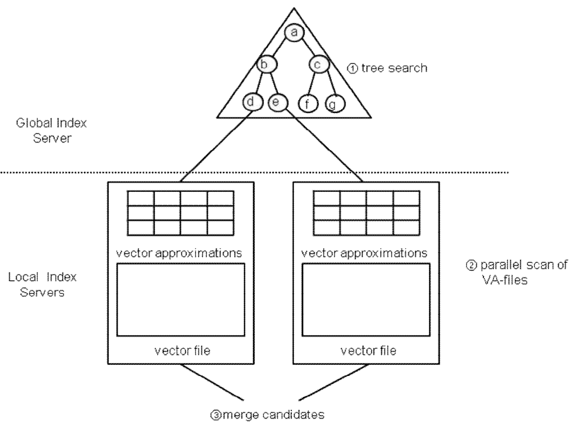

Figure 3. K-NN Search

The cost of k-NN query processing T_{QP} on a DVA-tree consists of the following components:

- Cost for traversing a hybrid spill-tree
- Cost for searching a k-NN query on a VA-file
- Cost for merging candidates

Relevant symbols and their descriptions are given in Table 1.

TABLE I. SUMMARY OF SYMBOLS AND RESPECTIVE DEFINITIONS

Symbol	Descriptions
D	number of dimensions
Q	query point
k	number of nearest neighbors
$\bar{k}$	average kth distance between points in a sample
$F(x)$	distance distribution
O_r	routing point stored in an internal node on a hybrid spill- tree
$r(N_r)$	covering radius of node N_r
l	number of nodes in a hybrid spill-tree
m	number of leaf nodes accessed for processing a range query on a hybrid spill-tree
v	number of points stored in a local index server
b	number of bits used for bit encoding (compressing)
t_{approx}	time to compute lower and upper bounds per dimension in a filtering step for a VA-file search
t_{vector}	time to compute the distance between two points per dimension
w	number of points remained after the filtering step of a VA-file search
t_{read}	time to load a block from a disk
$t_{compare}$	time to compare two distances between two points and a query point

For simplicity, we assume that data points are uniformly and independently distributed in the data space. First, consider a range query **rang**$(Q,\bar{k})$. A node N_r of the hybrid spill-tree has to be accessed iff the ball of radius $\bar{k}$ centered in the query object Q and the region associated with N_r interset. This is the case iff $d(Q, O_r) \le r(N_r) + \bar{k}$. For instance, the distribution of distance is $F(x) = Pr\{d(O_1, O_2) \le x\}$. The probability that N_r has to be accessed can be expressed [13] as

$$\begin{aligned}\Pr\{node\ N_r\ is\ accessed\} &= \Pr\{d(Q,O_r) \le r(N_r) + \bar{k}\} \\ &= F_Q(r(N_r) + \bar{k}) \approx F(r(N_r) + \bar{k})\end{aligned} \tag{8}$$

The expected number of nodes accessed for a range query is

$$nodes(range(Q,\bar{k})) = \sum_{i=1}^{l} F(r(N_{r_i}) + \bar{k}) \tag{9}$$

If the hybrid spill-tree fits entirely into main memory, no IO operation are necessary. Therefore, the average cost for a range query is the sum of the costs of distance computation among the query point and the accessed nodes.

$$T_{1st} = nodes(range(Q,\bar{k})) \cdot (D \cdot t_{vector}) \tag{10}$$

The local index servers corresponding to m leaf nodes determined by tree search process the k-NN query using a VA-file in parallel. The points v in each local index sever may be represented by a unique bit-string of length b. We consider the case that the approximation file fits into main memory. The cost of the filtering phase is

$$T_f = v \cdot D \cdot t_{approx}(b) \tag{11}$$

After the filtering phase, a small set of candidiates remain. In the refinement phase, the number w of points visited is represented in [14]. The disk IO occurs by random access to the vector file. The cost of the refinement step can be derived as

$$T_r = w \cdot (t_{read} + D \cdot t_{vector}) \tag{12}$$

Finally, the total cost of theVA-File based k-NN search is the sum of the costs of the two phases.

$$T_{2nd} = T_f + T_r \tag{13}$$

Each local index server returns k sorted candidate points. The final k nearest neighbors are determined by comparing $m{\cdot}k$ candidate points obtained from m local index servers. The merge cost of the candidate points is estimated as

$$T_{3rd} = k \cdot (m-1) \cdot t_{compare} \tag{14}$$

Finally, the estimated total cost for k-NN query processing is

$$T_{QP} = T_{1st} + T_{2nd} + T_{3rd} \tag{15}$$

IV. Experimental Results

In this section, we present an experimental study to evaluate the performance of the DVA-tree. The performance is evaluated using the average execution time and accuracy of a k-NN search over 100 different queries. We compare the performance of the DVA-tree with that of the distributed hybrid spill-tree [8] because the distributed hybrid spill-tree is a recent indexing structure based on a cluster environment.

The distributed hybrid spill-tree and DVA-tree algorithms were developed using the M-tree C++ package [15]. We report our experimental results based on real and synthetic datasets. We use a real data set, Aerial40 [16]. Aerial40 contains 270,000 feature vectors with 61 dimensions.

All the experiments were conducted on eight server machines in a Linux cluster based on a global file system. Each of the eight servers has a 3.40 GHz Pentium® D CPU processor with 2.4 GB of memory capacity. For the distributed hybrid spill-tree or DVA-tree, we dedicated a master server and six other servers as local index servers that execute k-NN queries either on the local hybrid spill-tree or VA-file. Meanwhile, we used the last server as a merger to integrate the k-NN search results from the local index servers. This is to construct a similar query execution environment as the MapReduce operations for the nearest neighbor search proposed in [8]. In order to emulate a larger configuration including more than six local index servers, we also ran multiple local index servers on a single machine. The intercommunication between the master server and local index servers is done via TCP/IP.

For a fair performance comparison, the top trees of the DVA-tree and distributed hybrid spill-tree are built on same sample data, and all the indexing structures have the same number of index servers. The number of bits per dimension of approximation cell used in the VA-file is 8.

In many applications, data points are often correlated in different ways. We test the performance of the DVA-tree and the distributed hybrid spill-tree on the skewed dataset of Aerial40. For a fair performance comparison, the top trees of the DVA-tree and distributed hybrid spill-tree are built on same sample data.

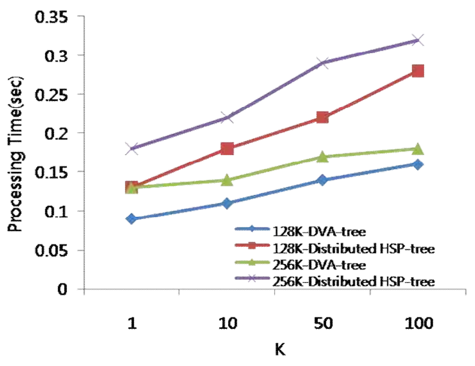

Figure 4. The search time on the skewed dataset.

Fig. 4 depicts the performance of the approximate k-NN searches as the number of the required nearest neighbors increases. The results show that the average execution time of the approximate k-NN searches on the DVA-tree runs up to 1.78 times faster than on the distributed hybrid spill-tree. Moreover, we can notice that the performance gap between the DVA-tree and the distributed hybrid spill-tree steadily widens as the number of the nearest neighbors increases. This is based on the fact that the DVA-tree executes the nearest neighbor search based on the VA-file, which scans the entire approximation data regardless of the number of required neighbors and performs disk operations for few vector data. However, the distributed hybrid spill-tree has an amount of overhead for processing directories of the tree, and this overhead increases when increasing the number of desired nearest neighbors. Therefore, the processing delay for a nearest neighbor search increases more slowly for the DVA-tree than for the distributed hybrid spill-tree.

On the other hand, both the DVA-tree and distributed hybrid spill-tree yield better performances, when we use the smaller page capacity of leaf nodes in the top tree. This can be explained by the fact that the top tree with smaller page capacity of leaf nodes enables the parallel k-NN queries to be performed over more local index servers. We observe that the DVA-tree yields better performance than the distributed hybrid spill-tree regardless of the size of the leaf pages in the top-tree. The results are shown in Fig. 4.

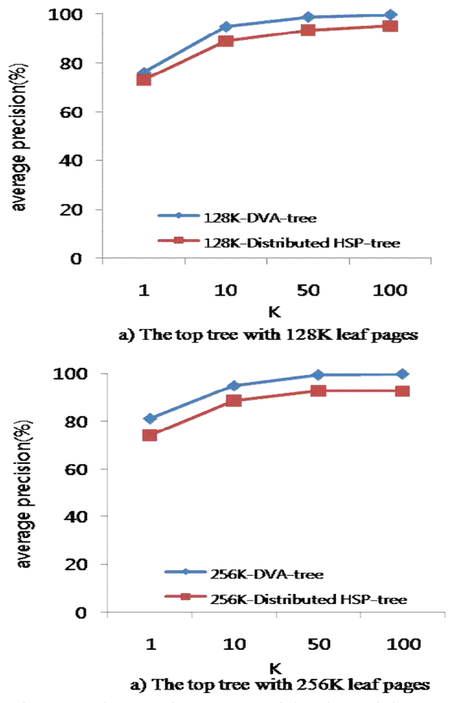

Figure 5. The search accuracy of the skewed dataset.

Fig. 5 shows the search accuracy by varying the page capacity of the tree from 128 KBytes to 256 KBytes. The DVA-tree obtains a better search accuracy compared to the distributed hybrid spill-tree when using the same sized leaf pages, because it performs k-NN queries based on the VA-file, which provides an exact k-NN search.

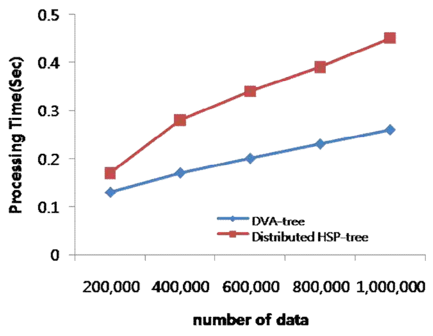

Figure 6. The k-NN search time for different data size.

Fig. 6 shows the performance of the approximate 100-NN search by varying the number of data from 200,000 to 1,000,000. This result is similar to those of experiments using the real dataset and clearly shows the effectiveness of the DVA-tree. The DVA-tree outperforms the distributed hybrid spill-tree in terms of execution time as the number of data increases. This is due to the fact that local index servers in the DVA-tree utilize the VA-file technique without any processing overhead of the directory of the tree. In recognition of this fact, if we consider a larger dataset or a higher number of dimensions, such as 100, the difference between search performances will widen even more.

V. CONCLUSIONS

In this paper, we presented the design of a new high-dimensional indexing scheme, called a DVA-tree, to solve the distributed k-nearest neighbor search problem over large-scale high-dimensional data in cluster environments. The DVA-tree employs a hierarchical clustering method and distributed VA-file management in order to allow a parallel k-NN search on each of the VA-files. We use a hybrid spill-tree as a clustering method and build the hybrid spill-tree on the sample data of large-scale high-dimensional data, because the sampling is independent of the dimensionality and the sampled data maintain the cluster information of the data set stored in the database. The data sets clustered by the hybrid spill-tree are managed on distributed VA-files. We proposed an algorithm for approximate k-NN searches over multiple machines. Our experimental evaluation indicates that the DVA-tree can efficiently provide a k-NN search with high accuracy. Moreover, since our algorithms are very simple, they are appropriate for data sets of tremendous size or dimensions.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MKE/KEIT. [10038768, The Development of Supercomputing System for the Genome Analysis]

REFERENCES

[1] S. Blanas and V. Samoladas, “Contention-base performance evaluation of multidimensional range search in peer-to-peer networks”, Journal of Future Generation Computer System, Vol. 25, Iss.1, pp.100-108, 2009

[2] H. V. Jagadish, B. C. Ooi, Q. H. Vu, R. Zhang, and A. Zhou, “VBI-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes,” Proc. IEEE International Conference on Data Engineering (ICDE 06), p 34, 2006.

[3] M. Bawa, T. Condie, and P. Ganesan, “LSH forest: self-tunning indexes for similarity search,” Proc. International World Wide Web (WWW 05), pp. 353-366, 2005.

[4] C. Schmidt and M. Parashar, “Flexible information discovery in decentralized distributed systems,” Proc. IEEE International Symposium on High Performance Distributed Computing (HPDC 03), 2003.

[5] N. Koudas, C. Faloutsos, and I. Kamel, “Declustering spatial on multi-computer architecture,” Proc. International Conference on Extending Database Technology (EDBT 96), pp. 592-614, 1996.

[6] B. Schnitzer and S. T. Leutenegger, “Master-Client R-trees: a new parallel R-tree architecture,” Proc. International Conference on Scientific ad Statistical Database Management (SSDBM 99), pp. 68-77, 1999.

[7] A. Guttman, “R-trees: a dynamic index structure for spatial searching,” Proc. ACM SIGMOD International conference on Management of Data, pp. 47-57, 1984.

[8] T. Liu, C. Rosenberg, and H. A. Rowley, “Clustering billions of images with large scale nearest neighbor search,” Proc. IEEE Workshop on Application of Computer Vision (WACV 07), pp. 28-33, 2007.

[9] R. Weber, H.-J. Schek, and S. Blott, “A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces,” Proc. International Confernce on Very Large Databases (VLDB 98), pp. 194-205, 1998.

[10] R. Weber, K. Böhm, and H.-J. Schek, “Interative-time similarity search for large image collections using parallel VA-files,” Proc. IEEE International Conference on Data Engineering (ICDE 00), pp. 83-92, 2000.

[11] J. Chang, and A. Lee, “Parallel high-dimensional index structure for content-based information retrieval,” Proc. International Conference on Convergence Information Technology (ICCIT 08), pp. 101-106, 2008.

[12] P. Ciaccia, M. Patella, and P. Zezula, “M-tree: an efficient access method for similarity search in metric spaces,” Proc. International Conference on Very Large Database (VLDB 97), pp. 426-435, 1997.

[13] P. Ciaccia, M. Patella, and P. Zezula, “A cost model for similarity queries in metric spaces”, Proc. ACM SIGPODS conference, pp. 65-76, 1998.

[14] R. Weger, and K. Böhm, “Trading quality for time with nearest-neighbor search,” Technical report, Dept . of Computer Science, 1999.

[15] M-tree homepage, http://www-db.deis.unibo.it/research/Mtree , retrieved: March, 2012.

[16] Real data source website, retrieved: March, 2012 http://www.autolab.org/autoweb/15960.html