World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:4, No:9, 2010

Balancing Strategies for Parallel Content-based Data Retrieval Algorithms in a k-tree Structured Database

Radu Dobrescu, Matei Dobrescu, Daniela Hossu

***Abstract*—**The paper proposes a unified model for multimedia data retrieval which includes data representatives, content representatives, index structure, and search algorithms. The multimedia data are defined as k-dimensional signals indexed in a multidimensional k-tree structure. The benefits of using the k-tree unified model were demonstrated by running the data retrieval application on a six networked nodes test bed cluster. The tests were performed with two retrieval algorithms, one that allows parallel searching using a single feature, the second that performs a weighted cascade search for multiple features querying. The experiments show a significant reduction of retrieval time while maintaining the quality of results.

***Keywords*—**balancing strategies, multimedia databases, parallel processing, retrieval algorithms

I. Introduction

MULTIMEDIA databases are extending the scope of traditional databases to handle the complex structure of multimedia objects. Models for multimedia information must include representations for the structure and content of several media in a form that allows flexibility in retrieval. Data processing of the different types of multimedia signals exhibits the familiar trade-offs; one must decide whether data quality, storage requirement, or computation speed plays the crucial role. In this paper we have utilized the metadata model with three classes of objects (Description Units, Segments and Descriptors) introduced in [1]. A description unit (DU) defines an object or collection of objects that can be given a context in terms of creation and relationship with other objects. A segment captures the notion of some part of an object that can be independently analyzed in terms of content and be used by it, but it has no context of its own, getting it from the DU of the object it belongs to. A descriptor is a representation of a feature and a descriptor value is its instantiation. In terms of structure DU's, segments and descriptors are organized as part-of hierarchies. A DU is either a root unit or is related to the DU describing the collection of units where it belongs, which has its own DU. Each multimedia data type can be viewed as k-dimensional (k-d) data in spatial-temporal domain [2].

Each dimension of the data is separated into small blocks and then formed into a multidimensional tree structure, called a k-tree. Using the k-tree structure, the retrieval time improves while the retrieval accuracy remains relatively constant. Moreover, since we can realize all types of multimedia data using the same k-tree data structure, the data indexing and retrieval algorithms are uniform [3].

Radu Dobrescu is with the "Politechnica" University of Bucharest in Romania (e-mail: radud@isis.pub.ro).

Matei Dobrescu is with the "Politechnica" University of Bucharest in Romania .

Daniela Hossu is with the "Politechnica" University of Bucharest in Romania

II. A model for multimedia metadatabases

In the data model, an instance of DU is linked to an instance of a higher-level DU, in a many-to-one relationship. Metadata concerning object content is centered in the Segment and Descriptor classes. A segment may be linked to some DU providing its context. Several segments may be attached to the same DU, accounting for different parts of its content. A segment may instead be attached to another segment, providing a specialized description of one of its parts. Segments are further specialized for text, image and video and are linked to descriptors specific to each media. The metadata model has been translated to an operational database system where tables and attributes closely follow the proposed model. This relational prototype has been populated with 4 sets of data (referred later as multidimensional signals): descriptions of text documents, a collection of photos (still images), video sets and audio sets.

Metadata are data about data. Broadly speaking, metadata can refer to any data that are used to describe the content, quality, condition and other aspects of data for humans or machines to locate, access and understand the data. An image itself tells nothing more than the plain fact that it is a specific picture. Without reading the associated metadata, it is impossible for a user to know the properties of the image such as who took the picture, when and where was the picture taken, what is the resolution of the picture etc, all of which are important information that helps to determine the suitability of the image for a particular application before the user takes a look at the actual data. Metadata plays far more important role in managing multimedia data than does the management of traditional structured data. Some of the reasons are :

1) Different Query Paradigm The exact-match paradigm for querying is no longer suitable or adequate for querying or retrieving various types of digital data.

2) Inadequate Processing Technique Content-based processing techniques are too hard to analyze and very large data-sets are often limited or inadequate.

3) Lacking efficiency When a content-based search is possible, it cannot be used very frequently (e.g. for every query) due to performance reasons and because of varying application.

4) Semantics of multimedia data Derive and Interpreted data(which may be considered a part of metadata) as well as context and semantics (which may be easier to base on metadata rather than raw data) are of greater value when dealing with multimedia data.

One can classify Metadata in various categories, but two categories are essential:

i. Based on dependence on content

One can distinguish content-independent metadata (which captures information that does not depend on the content of the document with which it is associated, as for example location or date of document) and respectively content-dependent metadata that depends on the content of the document it is associated with. Example of content dependent metadata are size of a document, colors, number of rows and columns in an image.

ii. Based on hierarchical utilization

1) Administrative Metadata used in managing and administering information resources.

2) Descriptive Metadata used to describe or identify information resources.

3) Preservation Metadata related to the preservation management of information resource

4) Technical Metadata related to how a system functions or metadata behave

In the case of structured databases, the norm for the generation of Metadata is to use schema descriptions and associated information (such as database statistics) as metadata. In the case of unstructured textual data and information retrieval, metadata is generally limited to indexes and textual descriptions of data. Metadata in such cases provides a suitable basis for building the higher forms of information. Metadata is commonly generated via three methods: - Analyzing raw data; - Semi-automatic augmentation; - Processing with implicit metadata generation.

The *k*-tree, conceived to store *k*-dimensional points data, can be used to represent such metadata. Purpose is always to hierarchically decompose space into a relatively small number of cells such that no cell contains too many input objects. This provides a fast way to access any input object by position. We traverse down the hierarchy until we find the cell containing the object and then scan through the few objects in the cell to identify the right one. Typical algorithms construct *k*-trees by partitioning point sets. Each node in the tree is defined by a plane through one of the dimensions that partitions the set of points into left/right (or up/down) sets, each with half the points of the parent node. Partitioning stops after log(*n*) levels, with each point in its own leaf cell.

Content-based retrieval of multidimensional signals is done by comparing features extracted from the input query with features extracted from every record in the database. The features of a multidimensional signal are subjective information. They are characteristics that are used to distinguish one signal from others. A 2-dimensional signal, such as an image, is characterized by features such as color, texture, and intensity. The basic algorithms for the searching of data in each of the different domains are quite similar. In this paper, a unified retrieval framework based on both keyword annotations and visual features is proposed. In this framework, a set of statistical models are built based on visual features of a small set of manually labeled images to represent semantic concepts and used to propagate keywords to other unlabeled images. These models are updated periodically; in this sense, the keyword models serve the function of accumulation and memorization of knowledge learned from user-provided relevance feedback. A matching search requires that the index key (defining feature) be unique and matched to the query. Exactly matched searching requires exhaustive comparisons that are inefficient and unsuitable for multidimensional signals; similarity searching is more appropriate. A similarity-search re-orders the database by distance between each record and the query; the result is selected from the ranking. In order to improve the retrieval procedure, two sets of effective and efficient similarity measures and relevance feedback schemes are proposed for query by keyword scenario and query by image example scenario, respectively.

III. Design of a K-tree Database Model

A. Basic description of the k-tree concept

A k-tree [4] is a directed graph; each node has 2k incoming edges and one outgoing edge with a balanced structure. A tree expresses the hierarchical relationships of data between consecutive levels. The k-tree is used to store the k-dimensional data. For example in case of a quad tree, k=2 and the tree can hold a two-dimensional image. For a composite data type, the tree is more complex, each node containing more than one type of features. For example, a node of a tree derived from motion picture data is a composite node, which comprises of video and audio features.

A k-tree has three main benefits. First, the k-tree holds the features on the tree structure itself and so it reduces computation time to find distance to a comparison between two tree nodes. Second, a k-tree can accelerate multi-resolution processing by calculating small, global information first and then large, local information when precise resolution is needed. Third, the data of the k-tree is unified since only the degree of the tree changes, while the processing algorithm and data structure remain invariant. The structure of a k-tree is also a basis for parallel processing [5]. The reduction of

computation time is a function of how well the global data uniquely identifies the target.

B. Representation of k-tree features

To generate the features from data, a normalization technique is used. The domain of a feature is reduced to a set of selected values from the set of potential values. An identification number of each element in the reduced set is used. When data is inserted into the system, it is converted to the selected domain. The feature is represented by a histogram. To generate a feature k-tree, the salient features of the multimedia data are extracted to create the leaves of the k-tree. The features are summarized to create the next level of information; the most global information is stored at the root of the tree. For example, figure 1 shows the feature extraction of 2-dimensional data array into a quad-tree structure. Data is separated into blocked data and transformed into features, which are stored in the leaves of the tree. Information stored in the leaves is summarized in the next higher levels.

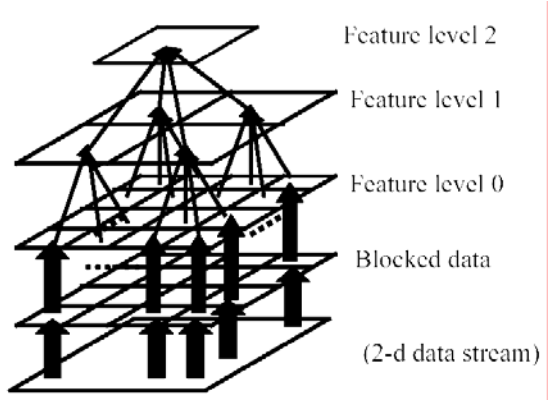

Fig. 1 Feature extraction of 2-dimensional k-tree

C. Mathematical definition of a k-tree structure

Let T denote the tree network under consideration, V denote the vertex set and E denote the edge set of T, respectively. Let $n = |V|$. The n vertices in V are labeled with 1, 2, and n, respectively. Denote *label(v)* as the label of a vertex $v \in V$. The tree network is undirected. Each edge $e \in E$ has an arbitrary positive length *w(e)*. A leaf of T is a vertex with degree one. Let m be the number of leaves of T. For any two vertices u and v in V, the distance between u and v, denoted by *d(u, v)*, is the length of the unique path connecting u and v. The distance from a vertex to a subtree is defined as the shortest distance from the vertex to any vertex in the subtree. We denote *d(v, X)* as the distance from a vertex v to a subtree X of T. The *distance-sum* of a subtree X of T, denoted by *Sum(X)*, is the total distance from X to the vertices of T. The *eccentricity* of a subtree X of T, denoted by *Eccen(X)*, is the distance from X to the farthest vertex of T. The *center* of T is any point of T whose eccentricity is minimum. A *k-tree core* of T is a minimum distance-sum subtree of T with exactly k leaves. A *k-tree center* of T is a minimum eccentricity subtree of T with exactly k leaves. Clearly, as a consequence of the minimum distance-sum criterion, a leaf of *k*-tree core shall be a leaf of T, but for a leaf of a *k*-tree center it is not necessary to be a leaf of T.

IV. Retrieval Algorithms on *K*-trees

A. Algorithm for finding the k-tree core of a tree T(V, E)

Let r be an endpoint of a two-tree core of T. We assume that T is rooted at r. For each vertex v in T, denote $p(v)$, *depth*(v), T_v, and *size*(v) as the parent of v, the depth of v, the subtree rooted at v, and the number of vertices contained in the subtree rooted at v, respectively. Let $P_{v,l}$ be the path from a vertex v to a leaf l of T_v. The *distance saving SV* of this path is defined as $SV(P_{v,l}) = \sum_{u \in V} d(u,v) - \sum_{u \in V} d(u, P_{v,l})$ and the *maximum distance saving MSV* as $MSV(v) = \max[SV(P_{v,l})]$. Let v be a vertex of T. Let l_1 and l_2 be two leaves of the subtree T_v. We say that l_1 *beats* l_2 at the vertex v if and only if: $SV(P_{v,l_1}) \le SV(P_{v,l_2})$

We say that a leaf l of T_v *dominates* T_v if and only if no leaf in T_v beats l at v. Denote $DL(v)$ as the leaf dominating T_v. By definition, $SV(P_v, DL(v)) = MSV(v)$.

Let l be a leaf in T and the path $P_{r,l}$ be given as $(u_1, u_2, ..., u_t)$, where $u_1 = r$ and $u_t = l$. There exists a vertex u_q, $1 \le q \le t$, such that the subtrees $T_{u_q}, T_{u_{q+1}}, \ldots, T_{u_t}$ are dominated by l but the other subtrees $T_{u_1}, T_{u_2}, \ldots, T_{u_{q-1}}$ are not. Denote $DE(l)$ as the vertex u_q. $DP(l)$ as the *dominated path* of l, which is the path from $DE(l)$ to l. Note that for any two leaves l_1 and l_2 in T, $DP(l_1) \cap DP(l_2) = \Phi$.

As an illustrative example, let us consider the tree network depicted in Fig. 2. In the figure, each edge of the tree network has a length 1 and the vertex u_1 is the root, which is an endpoint of a two-tree core of the network. The dominated paths of the leaves of the network are depicted with dashed lines. For each vertex u_i in the network, the values of *size*(u_i), *MSV*(u_i), *DL*(u_i), and *DE*(u_i) are listed in Table 1.

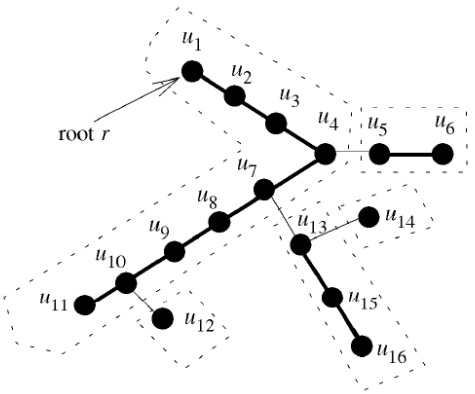

Fig. 2 The dominated paths in a *k*-tree core

TABLE I CHARACTERISTICS OF THE K-TREE CORE

u_i	$size(u_i)$	$MSV(u_i)$	$DL(u_i)$	$DE(u_i)$
u_1	16	65	u_{11}	-
u_2	15	50	u_{11}	-
u_3	14	36	u_{11}	-
u_4	13	23	u_{11}	-
u_5	2	2	u_6	-
u_6	1	1	u_6	u_5
u_7	10	13	u_{11}	-
u_8	5	8	u_{11}	-
u_9	4	4	u_{11}	-
u_{10}	3	1	u_{11}	-
u_{11}	1	0	u_{11}	u_1
u_{12}	1	0	u_{12}	u_{12}
u_{12}	4	3	u_{16}	-
u_{13}	1	0	u_{14}	u_{14}
u_{14}	2	1	u_{16}	-
u_{15}	1	0	u_{16}	u_{13}
u_{16}				

The algorithm to find the k-tree core has the following steps:

Step 1: Identify an endpoint r of a two-tree core of T, then orient T into a rooted tree with root r.

Step 2: For each vertex v in T, compute $size(v)$ and $depth(v)$.

Step 3: For each vertex v in T, compute $MSV(v)$ and $DL(v)$.

Step 4: For each leaf l in T, determine $R(l)$ and:

a. for each internal vertex v in T, broadcast $DL(v)$ to its children.

b. for each vertex v in T, if $v = r$, set $R(l) = MSV(v)$; otherwise, if $v \neq r$ and $DL(v) \neq DL(p(v))$, set $R(l) = w((p(v), v)) \times size(v) + MSV(v)$

Step 5: Find the leaf x in T with rank(x) = k - 1.

Step 6: Compute G_{k-1}, which is a k-tree core of T

a. for each vertex v in T, compute $A(v)$ as the largest number contained in the leaves of T_v.

b. for each vertex v, if $A(v) = 0$ remove v and the edge $(v, p(v))$.

c. Obtain G_{k-1} from T as follows: For each vertex v, if $A(v) = 0$ remove v and the edge $(v,p(v))$.

Step 7: Compute $Sum(G_{k-1})$ as $\sum_{v \in V} depth(v) - \sum_{l \in G_{k-1}} R(l)$

The parallel running time of the algorithm is discussed as follows: i) Steps 1 and 2 take $O(\log n)$ time using $O(n)$ work. Since $MSV(v)=max\{w((v, u))xsize(u)+ MSV(u)\}$, where u is a child of v, the computation of $MSV(v)$s is similar to that of the height of the subtree Tv, denoted $height(v)$. Thus, using tree contraction, $MSV(v)$s can be computed in $O(\log n)$ time using $O(n)$ work. ii) Clearly, during the computation of $MSV(v)$s, we can maintain some information to produce the values of $DL(v)$s in Step 3. Therefore, this step can be done in $O(\log n)$ time using $O(n)$ work. iii) The broadcasting performed in Substep 4a can be done in $O(\log n)$ time using $O(n)$ work. After the broadcasting, each vertex $v \neq r$ has the values of $DL(v)$ and $DL(p(v))$. Thus, Substep 4.b can be performed in $O(1)$ time using $O(n)$ work. iv) Step 5 is the most critical step, but one can consider that the $(k-1)^{th}$ element of a set of m - 1 elements can be determined in $O(\log m \log^* m)$ time using $O(m)$ work. v) Substeps 6a and 6c can be done in $O(\log n)$ time using $O(n)$ work. By tree contraction, Substep 6b can be easily done in $O(\log n)$ time using $O(n)$ work. vi) Step 7 can be done in $O(\log n)$ time using $O(n)$ work.

As a conclusion, except Step 5, all steps in the algorithm k-Tree_core can be implemented in $O(\log n)$ time using a total of $O(n)$ work.

B. Retrieval algorithm on k-tree based on similarity search

The basis of this algorithm is a function that measures the distance between the target and each of the multimedia objects and the reports the best matching, which are the data that have minimal distance between itself and the query. Figure 3 shows a block diagram of a similarity search, where N processing units $(P_0, ..., P_N)$ work on N partitions $(D_0, ..., D_N)$ of the database D.

The main problems of similarity search are the long computation requirement and the efficiency of index. First, because the similarity search bases on finding the minimal distances among all records, the larger is the database, the longer computation time is. Second, retrieval from a multimedia database is based upon similarity searches of the index space. A multimedia index entry should contain the salient features, which have been extracted from the raw data and the spatial-temporal relationships as a single entry. The usefulness of an index entry is a function of the computation time used to extract the features that define the entry, the space required to store the entry, and the ease with which the entity identifies the data.

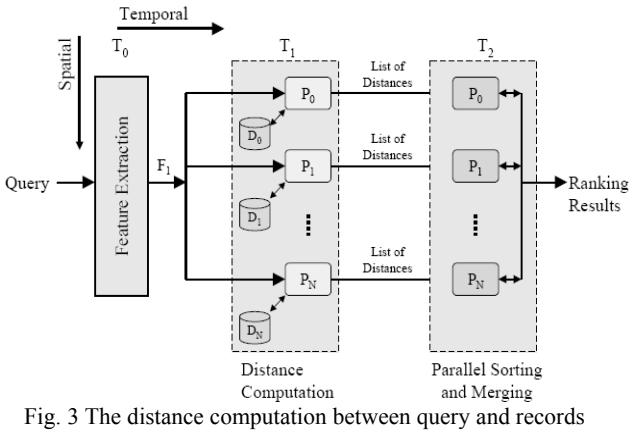

Fig. 3 The distance computation between query and records

Data comparison is made upon the distance of two objects. Let consider, in the general case that $A=a_1,a_2,...,a_n$ is the

feature list that defines an entity in a multimedia database; $Q=q_1,q_2,...,q_n$ is the list of features defining a query; $d_i(a_i,q_i)$ is the distance between the i^{th} feature of A and Q; w_i is the weight of the distance $di(ai,qi)$ where $\sum_{i=1}^{n} w_i = 1$. The total distance is given by $d(A,Q) = \sum_{i=1}^{n} w_i d_i(a_i, q_i)$.

Regular weighted comparison can be done by calculating distances of all features and then combining the weighted distances for final distance.

We can obtain a faster algorithm which can be processed in parallel if we proceed to a cascade weighted comparison, which is based on the concept that each feature is compared in sequence and that each feature has a relative importance. The ranking created by one feature determines the input for the next feature comparison. A weighting function is used to reflect the importance of the features. The features are processed in the order of their weights. The features with a lower weight use a subset of the total database determined by the higher weight features. The subset is the portion of the database that has the shortest distances to the query as determined by the higher weighted features. We generalize the distance as follows: suppose d(A,Q)=D1…k(A,Q) is a combined distance among all features from 1 to k.

The cascade distance is defined by $D_{1...n}(A,Q)=w_n d(a_n,q_n)+D_{1...(n-1)}(A,Q)$.

C. A generalized algorithm for virtual-node comparison

This algorithm allows to calculate the feature distances between the root of the query tree and the roots of all data trees in the database, and to select those data types for which the distances are below a threshold value. These candidates will be searched at a higher-resolution level in the next step. We can consider three situations:

Case a) the query's tree aligns within the *k*-tree structure of data.

1) Find the feature distances between feature in root of query tree and nodes of data at level L_{i-1} (nodes with solid-line link in Fig. 4) of the stored data. If the distance to a child node is equal to that between the query and its parent (L_i), the query may be found within that child node.

2) Repeat Case a) recursively on this child node. If there is no distance at level L_{i-1} close to the distance to the parent, the query is "not aligned"

Case b) the query data falls in between two or more nodes

1) If no node in *k*-tree (darker nodes in Fig. 4) can be a candidate, virtual nodes between two nodes have to be generated from the parts of their child nodes.

2) Repeat the whole algorithm into a new tree

Case c) the height of query is equal to a node height.

1) Search by Thumbnail (search data that is similar with disregarding to the scale of the query) with histogram quadratic distance to calculate the distance and then

2) Return the distance.

.

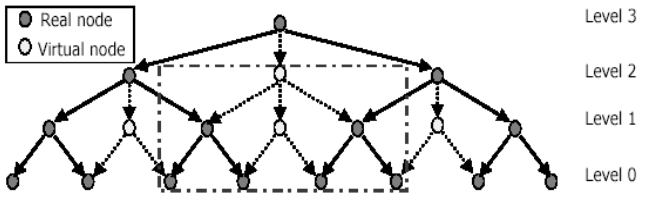

Fig. 4 Virtual-node structure for k=1

V. Experimental procedures

A. Algorithms for parallel processing

The first implemented algorithm allows parallel searching using a single feature. Prior to the search the database is distributed among the processors. Each processor performs the comparison between the query and its partition of the database. The results are sorted in parallel creating the final ranking. The second algorithm performs a weighted cascade search for multiple features querying. In this situation all distance computation blocks are replaced with the parallel model from a single feature query.

The software architecture necessary to perform parallel processing has three main components. The first component contains routines for database partitioning, in order to indicate which data parts should be processed by each processing unit.

The second component contains a large set of sequential operations typically used in content-based retrieval. Each operation that maps onto the functionality as provided by a generic algorithm is implemented by instantiating the generic algorithm with the proper parameters, including the function to be applied to the individual data elements. Parallel implementations of generic algorithms are obtained by inserting communication operations in the concatenation of sequential library routines.

The last component of the software architecture is the scheduling component that is applied to find an optimal solution for a given application. The requests for scheduling results are performed to determine which parallelization strategy is required. The aim of scheduling is to provide specified shares of the total system capacity to groups of jobs [6].

B. The cluster test-bed

In order to create a parallel/distributed environment we have built a cluster using commodity hardware and running Linux as operating system [7]. The cluster can run applications using a parallelization environment [8]. We have tested writing and running applications using PVM (Parallel Virtual Machine) which is a framework consisting of a number of software packages that accomplish the task of creating a single machine that spans across multiple CPU's, by using the network inter-connection and a specific library. Applications must be compiled using this specific library in order to permit communication. Another framework that can be used to run applications in a distributed manner is MPI

(Message Passing Interface). MPI specifies a library for communication between tasks [9].

Our cluster consists of a "head" machine and a number of six cluster nodes. The "head" provides all services for the cluster nodes – IP allocation, booting services, File System (NFS) for storage of data, facilities for updating, managing and controlling the images used by the cluster nodes as well as access to the cluster. The "head" computer provides an image for the operating system that is loaded by each of the cluster nodes since the cluster nodes do not have their own storage media. As this image resides in the memory of each cluster node, we took special steps to reduce the size of this image and to make most of the memory available to the running processes. We were able to reduce this image to 16 megabytes by moving different parts of a running Debian Linux system over network file systems, leaving on the image only those components needed for booting and controlling the cluster nodes.

The application partition is mounted read-only while the partition where data is stored is mounted read-write and accessible to the users on all machines in a similar manner providing transparent access to user data. In order to access the cluster, users must connect to a virtual server located on a head machine. This virtual server can also act as a node in the cluster when extra computation power is needed for the single streams.

C. Criteria to choose a load balancing strategy

Load balancing is defined as the allocation of the work of a single application to processors at run-time so that the execution time of the application is minimized. Since the speed at which a parallel application can be completed depends on the computation time of the slowest workstation, efficient load balancing can clearly provide major performance benefits. The two major categories for load-balancing algorithms are static and dynamic.

Static Load Balancing. Static load balancing algorithms allocate the tasks of a parallel program to workstations based on either the load at the time nodes are allocated to some task, or based on an average load of our workstation cluster. The advantage in this sort of algorithm is the simplicity in terms of both implementation as well as overhead, since there is no need to constantly monitor the workstations for performance statistics. However, static algorithms only work well when there is not much variation in the load on the workstations.

Dynamic Load Balancing. Dynamic load balancing algorithms make changes to the distribution of work among workstations at run-time; they use current or recent load information when making distribution decisions. As a result, dynamic load balancing algorithms can provide a significant improvement in performance over static algorithms. However, this comes at the additional cost of collecting and maintaining load information, so it is important to keep these overheads within reasonable limits.

Load Balancing Algorithm. While many different load balancing algorithms have been proposed, there are four basic steps that nearly all algorithms have in common:

1) Monitoring workstation performance (load monitoring)
2) Exchanging this information between workstations (synchronization)
3) Calculating new distributions (rebalancing criteria)
4) Actual data movement (job migration)

Balancing Strategies. There are three major parameters which usually define the strategy a specific load balancing algorithm will employ. These three parameters answer three important questions: i) who makes the load balancing decision, ii) what information is used to make the load balancing decision, and iii) where the load balancing decision is made. Global or local policies answer the question of what information will be used to make a load balancing decision. In global policies, the load balancer uses the performance profiles of all available workstations. In local policies workstations are partitioned into different groups. The benefit in a local scheme is that performance profile information is only exchanged within the group. The choice of a global or local policy depends on the behaviour an application will exhibit. For global schemes, balanced load convergence is faster compared to a local scheme since all workstations are considered at the same time. However, this requires additional communication and synchronization between the various workstations; the local schemes minimize this extra overhead. But the reduced synchronization between workstations is also a downfall of the local schemes if the various groups exhibit major differences in performance.

Centralized vs. Distributed Strategies. A load balancer is categorized as either centralized or distributed, both of which define where load balancing decisions are made. In a centralized scheme, the load balancer is located on one master workstation node and all decisions are made there. In a distributed scheme, the load balancer is replicated on all workstations. Once again, there are tradeoffs associated with choosing one location scheme over the other. For centralized schemes, the reliance on one central point of balancing control could limit future scalability. Additionally, the central scheme also requires an "all-to-one" exchange of profile information from workstations to the balancer as well as a "one-to-all" exchange of distribution instructions from the balancer to the workstations.

The implemented load balancer. We have implemented a Single Program Multiple Data (SPMD) computation algorithm, which implies that all the workstations run the same code, but operate on different sets of data. Each task is divided into operations. Workstations execute the same operation asynchronously, using data available in the workstation's own local memory. This is followed by a data exchange phase where information can be exchanged between workstations (if required), after which all workstations wait for synchronization. Thus each lock-step of an SPMD program contains 3 phases: i) calculation phase (each task will

do the required computation; there is no communication between workstations at this point); ii) data distribution phase (each task will distribute the relevant data to other tasks that need it for the next lock-step); iii) synchronization phase (this phase ensures that all tasks have completed the same lock-step).

The algorithm uses the following parameters:

T_{comp-i} – the interval between the time at which the first task on workstation *i* starts execution and the time at which the last task on the same workstation completes the computation and waits for the synchronization.

T_{task-i} – the average computation time for each task on a workstation, defined as:

$T_{taski} = T_{comp-i} / n_i$ (equ. 1)

T_{high} – the maximum of T_{comp} over all workstations

T_{low} – the minimum of T_{comp} over all workstations

In the SPMD algorithm T_{task} can be used to update T_{comp}. Thus if *m* tasks are moved to

workstation *i*, we can solve equ.1 to give us an estimation of T_{comp-i}: T_{task-i} x $(n_i + m)$.

D. Tests and results

The data parallel search algorithm illustrated in Figure 3 were ran on the test-bed cluster, for a single feature (k=1) and for two features (k=2). In both cases two computing schemes were tested: the regular computing scheme and the cascade weighted computing scheme. In these tests, data in each storage D_i are parts of feature index, so prior to use the database is distributed to the processors in the cluster. At the first stage (T1), each processing unit performs a feature based comparison between the query and the data in its assigned records. The output from each processor is a list of distances. At the second stage (T2), the results are sorted in parallel to create the overall ranking of the database records. The results of one feature using the parallel search algorithm are the input of the next feature. Each processor works on different portion of the database. Then, the sorted results are combined to the final results. Table 2 presents the time (in seconds) for each algorithm and for a different number of processing unit. For rapid comparison, the same results are displayed in fig. 5.

TABLE II PROCESSING TIME FOR RETRIEVAL TESTS

Proc.	*K=1*		*K=2*	
units	regular	cascade	regular	cascade
1	420	268	585	410
2	282	221	352	269
4	196	143	220	185
6	132	116	198	161
8	112	104	170	144

In all experiments was used the distance between pairs of results from consecutive levels of a *k*-tree. For different types of data there was identified the limit k_l-level that does not improve the result, but requires a significantly longer processing time. Once this level was identified, the unnecessary levels were eliminated of the *k*-tree and so the processing time storage was reduced.

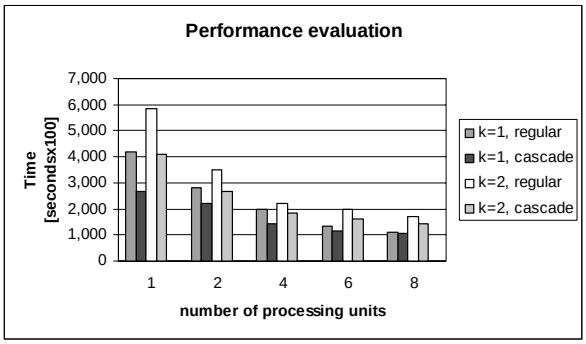

Fig. 5 Retrieval times versus number of processing units

Anyway, the computation time decreases significantly as the number of processors used to perform the computation increases. The achieved speedups are super-linear because each processor has more system resources available at the run-time; the smaller number of processors, the more frequent trashing occurred.

VI. Conclusions

We have introduced a parallel model for multimedia data retrieval by content with a multidimensional k-tree structure. The objective was to select the optimal level of a k–tree: the level furthest from the leaves that can distinguish the data structure. To achieve this goal the experimental research was concerned primarily to improve the efficiency of the search and consequently to reduce the response times of the algorithms. By exploiting the hierarchical nature of the trees, the search was improved by the use of multi-resolution algorithms using a weighted cascade. The experimental results show that multi-resolution processing can reduce retrieval time, maintain the accuracy, and exploit parallelism.

The model allows the extension of the system for the new types of data, new techniques, and new types of interest contents with less effort. In this aim we have proposed an algorithm to extract the k-tree core from a general features tree and to apply it in modeling a distributed database system.

Further work includes the validation of the proposed metadata model in a large database, the development of new criteria for searching on multimedia content and the use of the information gathered in the database. We intend to extend the parallelism into heterogeneous systems environments by using information on both task and machine characteristics in order to decide which processor a task should be allocated to.

Acknowledgement

This work was partially supported by the Romanian Ministry of Education and Research under Grant ID_1039, No. 121/2007.

REFERENCES

[1]Ribeiro, C., David, G., Calistru, C. (2004): A multimedia database workbench for content and context retrieval. *IEEE 6th Workshop on Multimedia Signal Processing* p.430-433

[2] Bradshaw, B. (2000): Semantic Based Image Retrieval: A Probabilistic Approach.*Proc. of ACM. Multimedia*,.23(6), p. 676—689

[3] R. Zhao and W. I. Grosky. (2002): Narrowing the semantic gap— improved textbased web document retrieval using visual features. *IEEE Transactions on MultiMedia*, vol. 4, no. 2 p.189–200

[4] Wang, B-F. (1998): Finding a k-Tree Core and a k-Tree Center of a Tree Network in Parallel. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, Issue 2. p.186-191

[5] Kiranyaz, S., Ferreira, M., Gabbouj, M. (2006): Automatic Object Extraction over Multi-Scale Edge Field for Multimedia Retrieval", *IEEE Transactions on Image Processing*, p.3759-3772

[6] Dobrescu, M. (2005): Distributed Image Processing Techniques for Multimedia Applications. *Ph.D. Thesis*. Politehnica Univ. of Bucharest

[7] Dobrescu, M., Mocanu, S. (2004): Resource management for real time parallel processing in a distributed system", *WSEAS Transactions on Computers*, Issue 3, vol.2, p.732-737

[8] Dobrescu, R., Dobrescu, M., Hossu, D. (2008): Fractal Analysis of Internet Traffic using a Parallel Processing Network Simulator, *Proceedings of the Sixth Int. Symp. Comm. Systems, Networks and Digital Signal Processing*, p. 622-625,

[9] Riley, G. , Ammar, M..Fujimoto, R., Park, A., Perumalla, K., Xu, D. (2004): A Federated Approach to Distributed Network Simulation. ACM Trans. on Modeling and Computer Simulation (TOMACS), Vol. 14(2)

Radu N. Dobrescu (M'90–SM'03) Born in 11.01.1946, in Braila, Romania. Dipl.Eng. degree in Automatic Control from the Faculty of Control and Computers of the Polytehnical Institute of Bucharest, in 1968. Ph.D. degree in Electrical Engineering from the Polytehnical Institute of Bucharest, Romania, in 1976.

He is currently Professor in the Department of Automation and Industrial Informatics of the Faculty of Control and Computers, "Politehnica" University of Bucharest, head of the laboratory on Data Transmission and Industrial Communication. From 1992 Ph.D. adviser in the field of Control Systems Engineering. Several scientific works in three main domains: Data acquisition, processing and transmission; Local area networks and industrial communication, Complexity and chaos theory

Prof. Dobrescu is a pioneer in fractal theory applications for medical image processing and biological systems modeling and simulation. Organizer of four International Symposiums on Interdisciplinary Approaches in Fractal Applications (IAFA 2003, IAFA 2005, IAFA 2007 and IAFA 2009).

Matei R.N. Dobrescu. (M'00) Born in 27.11.1976 in Bucharest, Romania. Dipl.Eng. degree in Automatic Control from the Faculty of Control and Computers of the "Politehnica" University of Bucharest, in 2000. Ph.D. degree in Systems Engineering from the "Politehnica" University of Bucharest, in 2004.

Currently Researcher in the Department of Automation and Industrial Informatics of the Faculty of Control and Computers, "Politehnica" University of Bucharest, engaged in a post-doctoral research stage. Several scientific works, especially multimedia applications with parallel and distributed processing:

Dr. Dobrescu has received in 2005 the Excellence Award Werner von Siemens for the best doctoral thesis of the year in the field of IT&C.

Daniela Gh. Hossu. (M'98) Born in 04.09.1971 in Craiova, Romania. Dipl.Eng. degree in Industrial Informatics from the Faculty of Power Engineering of the "Politehnica" University of Bucharest, in 2000. Ph.D. degree in Systems Engineering from the "Politehnica" University of Bucharest, in 2002.

She is currently Professor in the Department of Automation and Industrial Informatics of the Faculty of Control and Computers, POLITEHNICA University of Bucharest, head of the laboratory on Data Compression. She has several contributions in the non-standard control of power systems based on hybrid or neuro-fuzzy structures and in the field of image processing and compression.

Prof. Hossu was the director of three main projects in the field of Industrial Informatics granted by the Romanian Ministry of Research.