A Distributed Multimedia Data Management over the Grid

Kasturi Chatterjee, S. Masoud Sadjadi, and Shu-Ching Chen

Abstract In this chapter, we propose a distributed multimedia data management architecture, which can efficiently store and retrieve multimedia data across several nodes of a Grid environment. The main components of the proposed system comprises of a distributed multidimensional index structure, a distributed query manager handling content-based information retrievals and a load balancing technology. The proposed distributed query manager embeds the high-level semantic relationships among the multimedia data objects into the k-NN based similarity search, thus bridging the semantic gap and increasing the relevance of query results manifold. This research has two major usabilities. First, it models a web environment where each node of the Grid can be considered as the nodes or sources of data in the world-wide-web. This should help to investigate and understand the challenges and requirements of future search paradigms based on content of multimedia data rather than on text annotations, as used currently. Second, it provides the foundation to develop content-based information retrievals as a possible Grid service. Extensive experiments were conducted with varied data sizes and different number of distribution nodes. Encouraging results are obtained that makes this endeavor a potential architecture to manage complex multimedia data over a distributed environment.

1 Introduction

Grid computing can be described as a form of distributed computing which combines the power of several computing nodes of varied computing resources to execute one or more tasks collaboratively in a seamless and transparent manner without

Kasturi Chatterjee[1], S. Masoud Sadjadi[1], and Shu-Ching Chen[1,2]
[1]School of Computing and Information Sciences, Florida International University, Miami, FL, U.S.A.
[2]State Key Laboratory of Software Engineering, Wuhan University, 430072, China.
e-mail: kchat001@cs.fiu.edu, sadjadi@cs.fiu.edu, chens@cs.fiu.edu

any central control [12][13][14]. In the recent years, the popularity of Grid Computing has enabled experts from different scientific backgrounds to use its high computing power to execute computation intensive applications. Often these applications are data intensive like in protein folding, semiconductor manufacturing, and DNA sequence analysis. Such applications need a well defined data management within the distributed Grid environment.

There are basically two different approaches of designing a Data Grid: namely management of static data and supporting dynamic data sets. The first approach is also called Level 0 Data Grid [7]. It does not address data management issues as updates, transactions, integrations, etc., which are typical to data that changes with time. Basically, it addresses two fundamental issues: data access and meta data access. The data access provides managing, accessing and transferring data that is stored in the storage (typically as file systems). It essentially implements a storage system abstraction, by which the applications need not be aware of the specific low-level policies utilized in the data management. The meta-data service provides a mechanism for presenting and using the information about the data (stored in the files). Different categories of meta data can be used: namely content information of the file, data creation environment, and application-specific information related to the data. Apart from these two basic functionalities, Level 0 Data Grid provides some added services such as authorization and authentication, resource allocation, and performance evaluation. Level 1 [26] data Grids are for dynamic data sets and accommodates methods such as access, management, transaction and synchronization of data. To develop data Grids comparable in performance and robustness to the traditional data management techniques, features including indexing, querying, and transaction management. should be provided effectively. These features should be incorporated seamlessly along with features which are typical to Grid environment such as data regionalization, data synchronization, and load balancing.

Multimedia data is more complicated than traditional text-based data both in representations as well as in access mechanisms involved during their retrievals. Multimedia data is typically represented as multidimensional vectors of low-level features (e.g., colors, textures, objects, etc.). In addition to the low-level information contained in them, they have high-level semantic information attached. The relationship between the low-level features and the high-level semantic information is quite fuzzy and gives rise to the *semantic gap* issue. This is a typical problem area in all types of multimedia data retrievals and affects the relevance of query results negatively. Thus, any data management frameworks for multimedia data should be able to accommodate both these atypical characteristics of multimedia data: namely the multidimensional representation and the semantic information. Though multimedia data is more complex than traditional text-based data; they are popular media of communication due to their expressiveness. Thus, their presence and requirements in today's popular applications cannot be avoided. Hence, to enhance the usability of Grid environment, the underlying data Grid should be able to manage multimedia data effectively as well. But, since multimedia data is quite different from traditional text-based data, their management frameworks should also be different. For example, the index structure for multimedia data should be multidimensional as

opposed to the popular single dimensional index structures of text-based data. Additionally, since their information needs are different, the retrieval methodologies that the database system should support are different too. All these calls for a dedicated multimedia data management framework over the distributed environment of a Grid architecture.

The Internet can be considered as a distributed environment and can be simulated with a Grid architecture. Several popular applications such as social networks, collaborative tools, and search use multimedia data heavily. Thus a multimedia data management architecture for Grid environment can be considered as a prototype for investigating multimedia data management in the Internet. One specific application which can benefit immediately from such layout is *multimedia search*. Currently, the multimedia search is based on *keywords* or *annotations*. Such search paradigm limits the relevance of the search results manifold. Firstly, a single multimedia object (an image or a video) can have multiple high-level semantic meaning attached to it as the semantics vary with the perspective of the user who labels it. Thus, one keyword will be unable to capture the different aspects of the users' perspectives. Secondly, the multimedia data is represented and stored as multidimensional feature vectors. Thus, for a keyword-based search, during retrieving them from the underlying storage, a relationship need to be established between the low-level features and the high-level semantics (keywords with which they are expressed). This relationship is often fuzzy and there exist a gap between them, called the semantic gap. This affects the relevance of the query results and degrades the quality of the search results. The best approach is to introduce a content-based search paradigm for multimedia data which will be distributed over the Internet. Thus, a successful layout of a multimedia data management and content-based retrieval system over the Grid will be a potential solution for solving the problem of managing multimedia data over the Internet.

In this chapter, we lay down the framework for distributed multimedia data management over a Grid Computing environment. It comprises of two categories of components: firstly, components related to the multimedia data management like index structure, and query manager; and secondly, components related to the Grid architecture like automatic load balancing techniques, and replica management policies. These two sets of components should seamlessly communicate with one another so that the overall goal of achieving multimedia data management over a distributed Grid environment is achieved. A database management system is primarily composed of two major blocks: a robust storage and efficient well-rounded retrieval mechanisms. An index structure is the backbone of both and is the useful connection between them. Traditional single dimensional index structures such as B-Tree [1] cannot handle the multidimensional feature vectors that are used to represent the multimedia data. Though there are numerous multidimensional index structures such as those in [8][3] that can handle the multidimensional aspect of the multimedia data but they lack the capability to handle the high-level semantic relationships efficiently. In our earlier works, we proposed multidimensional index structures including AH-Tree [4], HAH-Tree [5] and GeM-Tree [6] designed for efficient management of multimedia data comprising of images and videos. In this

chapter, we extend the usability of multimedia index structures in a distributed Grid environment. We propose a distributed query management technique which embeds a content-based similarity search into a k-NN based algorithm in a distributed environment. Additionally, we introduce the high-level semantic relationship into the index structure and the subsequent query processing with a stochastic construct called Markov Model Mediator [25]. We also introduce Grid specific components including a load balancing manager and semantic relationships manager between Grid nodes to enable the proposed multimedia database framework to be used successfully in a Grid environment. Extensive experiments with varied data load and computation nodes are performed. The promising results demonstrate the usability of the proposed architecture and its potential extensibility.

The rest of the chapter is organized as follows. Section 2 presents a discussion on the related works in the field of distributed data management techniques and Data Grids. Section 3 lays down the overall framework of the proposed system and discusses the different components in details. Section 4 provides a detailed empirical study of the proposed system. Section 5 presents a brief conclusion and future direction of this research.

2 Related Work

In this section, we study the existing works on three important aspects: Distributed Multimedia Database Management Framework, Distributed Index Structures and Data Grids. Developing a successful distributed multimedia database management framework over the Grid environment is basically a seamless combination of all these different aspects. Thus, understanding the characteristics of each aspect helps us clearly define the capabilities that should be incorporated into the proposed architecture. Also, it helps us identify the limitations of each individual aspect when handling multimedia data in a distributed Grid environment. Thus, this survey of related works enable us to appreciate the necessity of an effective multimedia data management framework to be incorporated into a Grid architecture.

Distributed Multimedia Database Management Framework: Though there are some proposed architectures for distributed multimedia database systems such as [2][17], none of them discusses the intrinsic database components; for example, the index structures, and the query manager in the distributed environment. [17] proposes an object-oriented database with an object request broker (a brokering server). It uses specialized repository servers for storing different multimedia data types. Using specialized servers enables some query functionalities such as content-based retrievals, and optimized access to be allocated at the repository level rather than at the database level. Thus clearly, here the data storage is separated from the main database functionalities. Hence, different database tuning and optimization techniques depending on both the data stored as well as on the user accessibility including query optimization, query cost determination, and index structures cannot

be used seamlessly across the entire framework (since storage and database functionalities are separated). [2] also treats each multimedia data as an object and does not represent them as feature vectors. Hence, there is no well-defined index structure to facilitate efficient storage and retrieval based on the contents. The retrieval is done with object graphs where two levels of object graphs are used: namely local and central. Thus, the logical relationships among the multimedia data objects are captured but their relationship in terms of their content as well as storage strategies are not handled. Moreover, it doesn't propose any index structure, as robust as ones used in relational database systems, to be deployed in a distributed environment.

Distributed Index Structures: A replicated index structures for distributed data was proposed in [24]. It proposes a method called dPi-tree and is based on the Pi-tree [11]. The index structures are replicated in each location of the distributed environment without message passing schemes. Though the proposed index structure can be utilized in a distributed environment, it is not tailored to suit the requirements of complex multimedia data. Firstly, although theoretically it is supposed to be able to handle multidimensional data, complex containment issues can arise. Secondly, it is a space-based index structure and hence does not support similarity search (based on distance calculation) naturally (unlike distance-based index structures). Finally, content-based retrievals, typical for multimedia data, are not embedded in the search methodologies. Although [19] proposes a distributed search tree in a dynamic distributed environment, it uses an extended binary leaf search tree. This limits the usability of such approach for multidimensional data representation. Also, [18] proposes a lazy update method for B+ tree in a distributed environment. However, B+ tree is not a suitable candidate to handle multimedia data as it cannot handle multidimensional data effectively.

Data Grid: [7] discusses the various approaches to designing a Data Grid. It defines the requirements that a data Grid must satisfy and APIs necessary for its implementation. The design of the early data Grids was based on four major principles: mechanism neutrality, policy neutrality, compatibility with Grid infrastructure and uniformity of information infrastructure. The architecture is typically a two-layered structure, where the lower layer provides the data Grid specific services like those related to the storage system and to the meta-data repository. The upper layer consists of the high-level components such as the replica selection service, and replica management service. The storage system utilized in the proposed architecture are basically file structures and use GridFTP for data transfers. There are no database components such as index structures or query managers associated with the storage and meta-data repositories. [14] defines a virtual Data Grid that is capable of encompassing the expertise of large distributed diverse multidisciplinary communication. It proposes general abstractions for representing data and computation. Further, it lays down a virtual data schema and an architecture that develops techniques for representing and maintaining data on an Open Grid Service Architecture (OGSA). These architectures are specifically for static data and do not addresses issues such as data synchronizations, and transaction data policies. To enable these frameworks to support dynamic data, services such as data regionalization, data synchronization,

transactional management, data locality, event notification, and data load functions need to be introduced [26]. Additionally, data grids should have specific data distribution and data replication policies. For example, distribution approaches such as round-robin, Gaussian, random and Poisson can be used [26]. Data replication policy [22] is an important characteristics of a data Grid. The combination of the data distribution and the data replication policy defines the ability of a data region to support an application with minimum amount of data movement.

3 Overall Framework

Figure 1 presents the overall framework of the Distributed Multimedia Architecture over Grid. Each data node of the Grid is connected to the other nodes and has a multimedia database management system embedded in it. Each data node has a GridFTP server that takes care of the physical transfer of multimedia objects from one node to another. The data is basically stored in a data server. The multimedia database framework is divided into four major components: a multimedia interface, a core DBMS engine, a content-retrieval engine and a high-level relationship manager. These four components interact with one another to achieve the major functionalities including query, and update. The multimedia interface handles the users' requests and access the other three components to provide the information requested by the user. The core DBMS engine manages the functions related to the database that store the multimedia data. It is comprised of sub-components that are useful to designing a successful database system in a distributed Grid environment.

While components such as a transaction manager, and a query optimizer are the general components needed for a complete database engine design, components specific to a distributed environment such as an automatic load balancing system are also present. The content-retrieval engine houses the index structure and the access manager. The index structure along with the access manager handles the content-based retrieval queries. The index structure is a replicated multidimensional index structure which logically spans across the data nodes over the Grid. Thus, the index structure can be considered as a single unit organizing all the data that the entire Grid is comprised of. The high-level relationship manager maintains the semantic relationship among the multimedia data objects. It has three major sub-components: an affinity relationship metric, a local affinity update unit and a global affinity synchronization unit. The affinity relation metric basically captures and stores the high-level relationship between the multimedia data objects, based on the user access and feedback, while utilizing the Markov Model Mediator construct (discussed in details in Section 3.2.1). The local affinity update unit collects the user feedback and access patterns and updates the affinity relationship metric after specific time intervals. The global affinity synchronization unit updates the global affinity metrics based on the update of the local affinity metrics. The maintenance and use of the global affinity synchronization enables the users to issue queries transparently to the Grid without concerning themselves about the location and relationships of the

multimedia data. Additionally the data Grid may contain other components specific to the Grid: namely a replica manager designed specially to cater the typical needs of multimedia data and applications; a failure management component designed to detect the non-functioning of a particular node and how to share the load among the functioning nodes; etc.

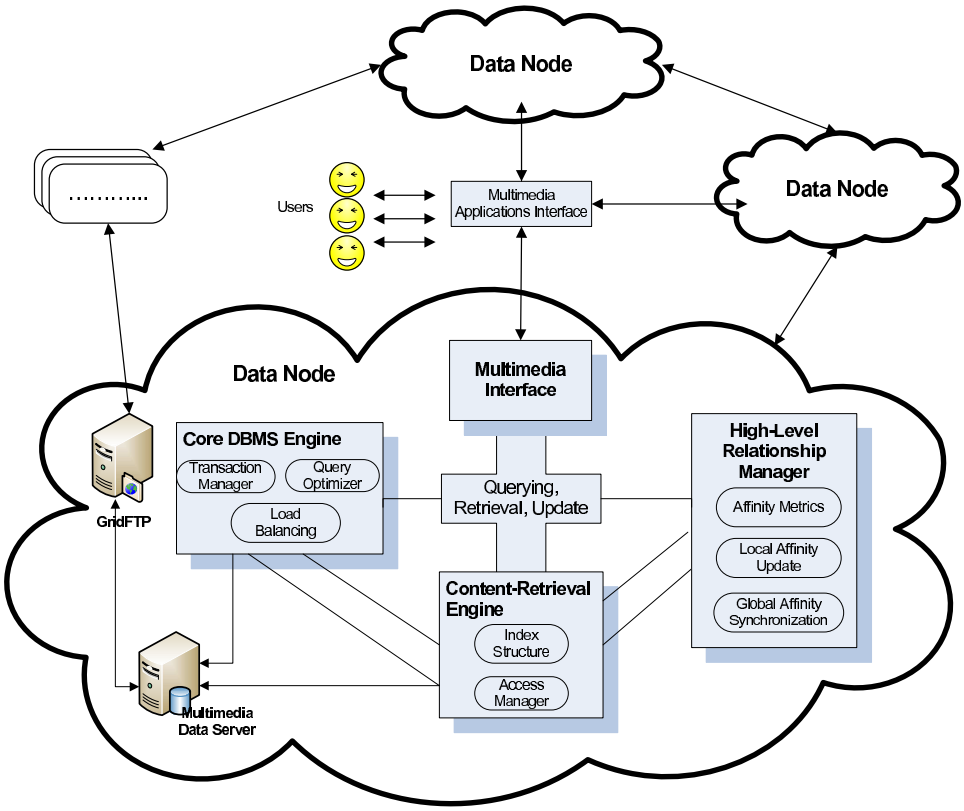

Fig. 1 Overview of the Proposed Framework.

3.1 Replicated Multidimensional Index Structure

As mentioned in Section 1, an index structure is the backbone of an efficient database management system and is the link between the data storage and the retrieval engines. For the proposed framework, the index structure should be designed to satisfy two basic requirements. First, it should be able to handle multimedia data efficiently; and second, it should be possible to be deployed over a Grid environ-

ment. To satisfy the first requirement, the index structure should be a multidimensional index structure, so that it can handle the multidimensional representation of the data objects. Also, the similarity search methods supported by the index structure should be able to handle the semantic relationship among the multimedia data objects along with the content-level closeness while answering the queries. To satisfy the second requirement, the index structure should be able to span across several distributed data locations and consider characteristics of each while dealing with user requests.

We proposed several multimedia index structure for different multimedia data types including images, and videos in our previous works [4][5][6]. [4] discusses a multidimensional index structure designed to handle images. [5] extends the idea to a hierarchical index structure, supporting video data objects. It can handle the hierarchical relationship among the different video units and facilitate intra and inter-unit retrieval strategies. Since having separate index structures for different multimedia data types can cause difficulty while embedding the index structure into the database kernel (both technical issues as well as optimization policy issues for other components residing in the kernel), [6] lays down a generalized index structure for managing both images and videos from one common platform. Additionally, it has the capability to be extended to support other forms of multimedia data such as documents. [4][5] and [6] support the popular multimedia data retrieval strategy based on content without violating the underlying indexed space. Basically, the k-NN algorithm, which is the standard similarity search algorithm for tree-based indexes, is customized to support the content-based retrievals while considering the high-level semantic relationships among the data objects. In this chapter, we extend the multidimensional index structure for images, called Affinity Hybrid Tree (AH-Tree) [4] and incorporate it into the proposed multimedia database management framework in a Grid environment. We chose AH-Tree as we wanted to use images as the test bed for developing and testing the initial framework of the distributed multimedia database. It should be pointed out here that both [5] as well as [6] can be used in the proposed framework without any loss of generality.

We use a replicated indexing approach, similar in philosophy to the one proposed in [24]. The multimedia data is distributed across multiple data nodes of a Grid and the index structure is replicated across multiple sites as well. Each data node with an index replica has efficient access to the local data. There is a logical link among the local index structures at each node. Thus while searching, the search results generated pick up the closest match to the submitted query among all the multimedia data present in the entire Grid repository. Each multimedia data is represented with a data signature that enables the system to uniquely identify a multimedia data object. The data signature F of a multimedia object is represented with two components, F_A and F_B.

$$F_A = \{x_1, x_2, ..., x_i\} \tag{1}$$

$$F_B = \{object_{id}, node_{id}, replica_{flag}\} \tag{2}$$

The feature vector representing the distribution of each multimedia data object is a union of the two parts represented as:

$$F = \{F_A \cup F_B\} \tag{3}$$

F_A represents the low-level feature vector of the multimedia data object and F_B is the unique identifier of the data object. The $object_{id}$ is the identifier of the multimedia data object, $node_{id}$ is the identification of the data node in the Grid where it belongs and $replica_{flag}$ is set to 1 or 0 depending on whether the particular data object has a duplicate entry in any of the other data nodes. If 1, the replica manager of the node under consideration should be consulted whenever this particular data is accessed or modified. The benefit of using the data signature is that it makes the proposed framework transparent to the type of multimedia data object used. Any multimedia data can be represented with F. F might need slight extension to capture the characteristics of the particular multimedia data used.

3.1.1 Node Structures

as a distance-based indexing, there are four basic node types as discussed in [4]: namely, Since AH-Tree is a hybrid structure with both a space-based indexing as well *Space_Index_Node*, the *Space_Data_Node*, the *Metric_Index_Node* and the *Metric_Data_Node*. The structure of the nodes of the AH-Tree is summarized in Table 1.

Table 1 Summary of Node Structures

	Node	Structure	Affinity Relationship
1	Space_Index_Node	dimension, split positions	X
2	Space_Data_Node	root node of metric index	X
3	Metric_Index_Node	root of subtree, extending radius, max affinity	$\surd$(max affinity)
4	Metric_Data_Node	indexed image objects	$\surd$

The leaf nodes (*Space_Data_Node* and *Metric_Data_Node*) of the index tree are linked with one another to enable easy sequential traversal. Also, a virtual link exists between the local index tree structures of the Data Grid, which have a large number of semantically related data objects. The *High-level Relationship Manager* along with the *Global Affinity Synchronization* component determines which data nodes (locations) of the Grid have large amount of semantically related data objects. Those index structures are logically linked to represent a virtual single multidimensional index structure.

3.1.2 Insertion and Deletion

To insert a node in the index structure, the tree is recursively traversed until a candidate leaf node is identified. A particular sub-tree leading to the leaf node is chosen by selecting an intermediate node for which there is no (as in Equation 4) or minimum increase (as in Equation 5) in the covering radius. Essentially, a new object is inserted at the leaf node, and if it is full, a split is required followed by a rearrangement of the tree with an increase in the number of levels. Thus, it can be seen that index structure grows in a bottom-up manner and hence maintains a balanced structure. Whenever a new data object is inserted into the index structure, an entry for its high-level semantic relationship with other multimedia objects is created in the *Local Affinity Update* component and the *Affinity Metrics*. As subsequent queries are issued, user feedback on the results generated are collected over time. They are used to populate/update values at *Affinity Metrics* and *Local Affinity Update* respectively.

$$d(O_r, O_n) \leq r(O_r) \tag{4}$$

$$d(O_r, O_n) - r(O_r) \rightarrow minimum. \tag{5}$$

To delete a node in the index structure, the tree is first traversed to locate the node. If it is an intermediate node, the pointer to the sub-tree it points to is set to zero and the memory is released. If it is a leaf-node, the actual data object at the repository pointed by it, is removed. As with any update, the *Local Affinity Update* component and the *Affinity Metrics* are modified to reflect the change.

3.2 Distributed Query Processing

The query processing component implements the most popular form of multimedia similarity search: namely, content-based retrieval. The Distributed Query Processing method is comprised of two major components. The first component is called the *Multimedia Application Interface* (as in Figure 1). It is a global query processing interface that takes in queries from the users and sends them across the data nodes of the Grid. At each data node, the queries are received by the local *Content-Retrieval Engine*, and is the second component of the Distributed Query Processor. The queries, once received by the individual local query processor are processed with the k-NN based similarity search algorithm of the multidimensional index structure. The k-NN algorithm (as discussed in Section 3.2.3) searches the underlying data repository based on both the low-level contents of the multimedia data and their high-level semantic relationship. The search results, comprising of the k closest data objects to the query, are returned from each data node of the Grid back to the *Multimedia Application Interface*. The search results, returned by each data node of the Grid have two pieces of information. First, the address of the multi-

media data object at the local repository of the particular Grid node; and second, its *distance* from the query object. The result sets from each data node of the Grid are merged together and sorted based on the *distance*. The top *k* objects from the sorted list are retrieved from their corresponding local repositories and form the final query result. Figure 2 demonstrates the distributed query process.

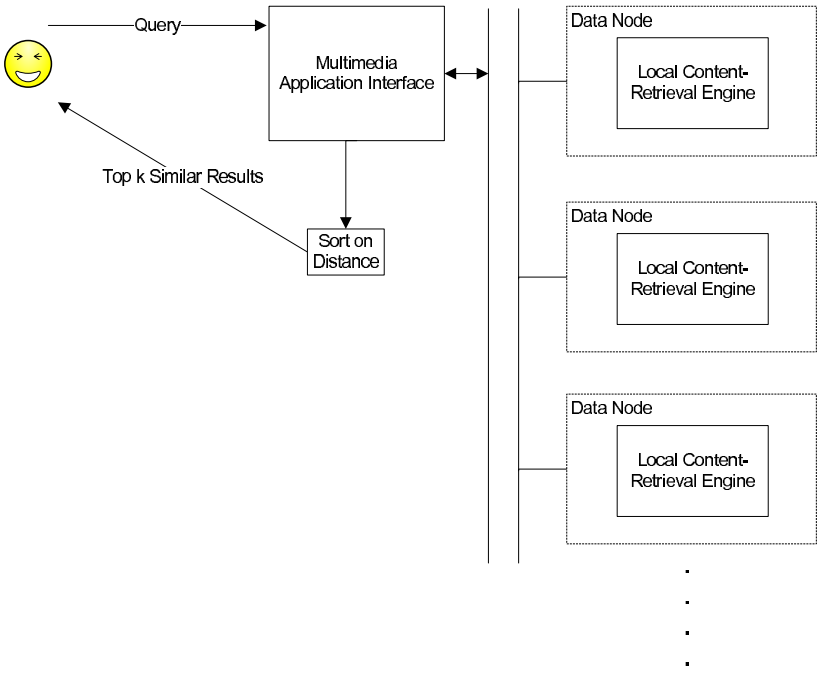

Fig. 2 Distributed Query Processing.

3.2.1 High Level Relationship

A major attribute for the successful processing of the issued queries is the efficient maintenance and use of the high-level semantic relationship among the multimedia data objects. There are three major components of the *High-Level Relationship Manager*: namely the *Affinity Metrics*, the *Local Affinity Update* and the *Global Affinity Synchronization*. The *Affinity Metrics* stores the affinity relationships (as discussed in Section 3.2.1) of the multimedia data objects stored in the local repository of the Grid node. The *Local Affinity Update* maintains the update information of the affinity values. The update process takes place whenever a new query is issued and the user feedback of the query results is obtained. The *Global Affinity Synchronization* helps in maintaining information necessary to synchronize the affinity relationship among the different Data Grids of the nodes.

For example, let Image # 101 and # 369 be marked similar by the user in a particular query instance. Also, let Image # 101 belongs to Node # 6 of the Grid and # 369 belongs to Node # 2. A *NXN* matrix (*N* is the number of nodes in the Grid) is updated at two locations (with same values): namely at the 6^{th} row and 2^{nd} column and 2^{nd} row and 6^{th} column and the affinity is increased between that particular pair of nodes.

If the number of nodes of a Grid are huge, it is not practical to store the semantic closeness among all the nodes of the Grid. Instead, semantic closeness between the Grid nodes belonging to logical regions are maintained. As mentioned earlier, in this chapter we use image as the testbed for the prototype framework. Thus, in the rest of the chapter, we discuss the different functionalities that handle images.

The high-level image relationship used in AH-Tree is captured using a stochastic construct called the Markov Model Mediator (MMM) [25], that maps the low level features and high level concepts in CBIR by capturing the image relationship as perceived by the user. MMM is a probabilistic based mechanism that adopts the Markov model framework and the mediator [25]. The MMM mechanism is represented as a 5-tuple $\lambda = (S, F, A, B, \pi)$, where S is the set of images, A is the state transition probability distribution, B is the feature vector and π is the initial state probability distribution. From this tuple, our point of interest is the state transition matrix denoted by A, where each entry *(i, j)* corresponds to the relationship between image i and j. The MMM mechanism builds an index vector for each image in the database and considers the relationship between the query image and the target image. The main idea is *the more frequent two images are accessed together, the more related they are*. The relative affinity measurement ($aff_{m,n}$) between two images m and n is defined as follows:

$$aff_{m,n} = \sum_{k=1}^{q} use_{m,k} \times use_{n,k} \times access_k \tag{6}$$

Here, $use_{m,k}$ denotes the usage pattern of image m with respect to query q_k per time period, and $access_k$ denotes the access frequency of query q_k per time period. The state transition probability matrix is built by having $a_{m,n}$ as the element in the $(m,n)^{th}$ position of A. The $a_{m,n}$ value is defined as

$$a_{m,n} = \frac{aff_{m,n}}{\sum_{n \varepsilon d} aff_{m,n}} \tag{7}$$

It should be noted that any high-level image relationship capturing mechanism similar to the affinity relationship can be used in the proposed index structure without loss of generality.

3.2.2 Affinity Promotion

As derived and proved in [4], the high-level semantic relationship cannot be incorporated into the multidimensional index structure as it violates the properties of the underlying indexed metric space. Thus, the affinity should be promoted from the leaf to the intermediate root level during each query. Initially, the affinity between the query object and the leaf nodes (at Level 0) of the index tree is determined. Then for each intermediate node at Level 1, the maximum of the affinity values of its children is calculated. This value is set as the affinity value of the particular intermediate index node at Level 1. The process continues for each Level till the root is reached. The affinity promotion technique has two important significances. First, it ensures that there is no false dismissal (i.e., if there is a candidate multimedia data object at some sub-tree of a node, the node and subsequently the sub-tree will be traversed). Second, it avoids unnecessary traversal of sub-trees where there is no possibility of the existence of any candidate node.

3.2.3 Distributed Content-Based k-NN Similarity Search

Table 2 presents the k-NN similarity search algorithm in a distributed environment that supports Content-Based Image Retrievals (CBIR) over Grid. It follows a branch and bound technique as in [16]. The algorithm presented in Table 2 is for the metric region. Before ensuing the search on the metric region, a filtering stage is undertaken where the space-based index structures in each node of the Grid is searched to get the k closest feature-spaces. They are merged together and the metric search is executed on them. Although every index structure can have two basic similarity search paradigms: namely Range Search and k-NN Search, for CBIR based retrievals, k-NN approach models the information requirements most naturally. Hence, we concentrate exclusively on the k-NN based search in this chapter. To issue content-based retrieval queries, a user must submit the query to the *Multimedia Application Interface*. The low-level features are extracted from it to represent the submitted query in the same feature space as that of the indexed data. For example, if the images stored in the Grid are represented as color and texture features, when a query image is submitted, it should be also represented as a feature vector comprising of color and texture features. The query in the form of the feature vector is submitted to the nodes of the Grid to the local multimedia interface at each Grid node. The affinity value is promoted in the multidimensional index structure as explained in Section 3.2.2. The index structure is traversed from the root to the leaf level. For each intermediate node of the index structure, the similarity between the indexed multimedia object and the query is determined in terms of both the low-level feature similarity and high-level semantic closeness. If the indexed multimedia object under consideration is more similar than the current k^{th} candidate in the priority list, it is replaced with the indexed multimedia object just considered. The priority queue is updated and the search continues recursively on the next closest candidate. Typically, the sub-tree contained in the candidate intermediate index entry is searched recursively.

If the examined node is a leaf and satisfies the similarity conditions of distance and affinity, the corresponding data object is pushed into the result set. The result set itself is another priority queue, where the results are prioritized according to the distance and affinity score with the query object. Once, each Grid node has a result-set ready, the result-sets are sent back to the *Multimedia Application Interface*. Here, the result sets get merged and sorted. The top k objects are returned to the user as the query result. The user feedback is collected on each presented query result and components in the High-Level Relationship Manager are updated accordingly.

When the number of Grid nodes is large, it is not practical to involve all the nodes for every query. Generally, under those circumstances, initially, the query is submitted to a *reasonable* number of Grid nodes (eg. in the range between $100 - 200$). After receiving the query results for the first iteration, the *Global Affinity Synchronization* of the Grid nodes, which have data objects marked similar to the query object by the user, is consulted. The Grid nodes that are most similar to the Grid node under consideration (i.e., those that contain similar multimedia data objects) are selected. In the next iteration, the refined query is submitted to these selected Grid nodes and the process continues. The merged and sorted result set produced at the end of each query iteration is stored. After a few iterations (the number of iterations depends upon the Grid layout), all the result sets are merged and sorted again to get the top k results corresponding to the issued query across the entire Grid multimedia data repository.

It should be pointed out here that to reduce the number of distance computations and use as many pre-computed distances as possible, a technique similar to [8] is introduced. In this method, in order to avoid unnecessary computing of distances between every pair of index entry with the query, the covering radius of a parent node, its distance with the child, along with its distance with the query object, is tested before a particular sub-tree is considered. It uses the classic metric space property of *triangular inequality* to formulate the checking condition. To reach a child node, its parent must have been traversed and thus there has to be a distance computation between the parent with the query. This distance computation is saved and reused for the next iteration. For example, one needs to start by computing the distance between the root with the query object. It then checks if any child of the root satisfies the qualification condition. If so, the corresponding child, along with its sub-tree, is considered.

3.3 Automatic Load Balancing

Any application in a Grid environment is incomplete without a proper *load balancing* functionality. Additionally, the domain that is dealt in this research (i.e. Multimedia Data) has an undeniable necessity for an effective load balancing component. This is because, multimedia data is much bulkier than ordinary text-based alphanumeric data and the *quality of service* expected from multimedia applications is much higher than traditional text-based retrieval methods. Thus, whenever a partic-

Table 2 Implementation of Distributed Content-Based k-NN Similarity Search

```
Distributed_Similarity_Search(Q, N_child, r(Q), aff) { //CBIR over Grid.
    Get User Query;
    Extract the low-level feature values from the query;
    Submit the query across the Grid;
    For each Node of the Grid do: {
     Affinity_Promotion( ); //promotion of affinity value.
     ∀ O_r in N_child do: {
        if (O_r is an intermediate index node) {
         if (| d(O_M, Q) - d(O_r, O_M)| ≤ r(Q)+r(O_r)) {
           Compute d(O_r, Q) and aff(O_r, Q);
           if ((d(O_r, Q) ≤ r(Q)+r(O_r)) && (aff(O_r, Q) ≥ aff)) {
             Distributed_Similarity_Search(ptr(T(O_r)), Q, aff);
                   //T(O_r): pointer to the subtree.
           }}}
       elseif (O_r is a leaf object){
         If the object qualifies the distance function and the affinity,
         add to the result set along with the distance d;
       }}}
     Merge result set from each Grid node;
     Sort result set on distance (similarity) with the query Q;
     Pick the k closest multimedia objects from the sorted result set;
}
```

ular Grid node is overloaded, the load should be distributed among the less-utilized Grid nodes to attain a balanced computation cost. Moreover, as discussed in Section 3.2.3, when a query is issued to a Grid, it is simultaneously issued to the Grid nodes. Query results from all the nodes of the Grid are collected and compiled to present the user with a single result set. Thus, if one/more node of the Grid is overloaded, it affects the performance of the entire Grid framework as the *Multimedia Application Interface* need to wait till it receives responses from all the Grid nodes. We note that in some applications, although load balancing may result in a more balanced utilization of resources, it may however worsen the overall performance. For typical Multimedia Data Application, this is not the case though.

We propose a load balancing algorithm as presented in Table 3. The basic heuristics used behind the proposed algorithm is computation_time ∞ number of indexed data points. Since, for developing the index structure and for subsequent queries, distances between pairs of multimedia objects need to be calculated. The number of necessary distance computations increases as the number of data objects involved increase. Now, the total number of distances computed determines the overall computation time. So, to balance the computation_time over the Grid, the number of multimedia data objects in each Grid node repository is balanced. The load balancing is typically not achieved in a single iteration but requires quite a few iterations. The number of iterations required depends on the data set involved. For each iteration, the maximum and minimum computation time for processing the submitted query is determined. Additionally, the Grid nodes having the maximum and minimum values, are identified. Normally, the number of data points in the Grid node

Table 3 Load Balancing in the Distributed Multimedia Database Management Framework

```
Load Balancing(n, i) { //Load Balancing over Grid.
  For each iteration i {
    Set min_time = minimum computation time among n Grids in iteration i − 1;
    Set max_time = maximum computation time among n Grids in iteration i − 1;
    Set n_min = node with minimum computation time;
    Set n_max = node with maximum computation time;
    if (number of data objects in n_max ≥ number of data objects in n_min)
      Set num_data_moved = (number of data objects in n_max − number of data objects in n_min)/2;
    else
      Set num_data_moved = x; //x is a pre-determined value.
    Move num_data_moved from n_max to n_min;
  }
}
```

taking the maximum time should be more than that taking the minimum time. If the condition is not as it is predicted, it can be concluded that the imbalance is not due to the query application but due to some other applications on the Grid. Under such circumstance a pre-defined number of data points are moved from the most loaded node to the least loaded one. The pre-defined number (x) is determined based on the initial load in each Grid node. If the condition is satisfied, data points are moved from the most loaded to the least loaded such that they both end up having the same data load. The process is repeated until a desired balanced state is reached.

It should be mentioned here that the proposed algorithm is devised with the assumption that the Grid under consideration is a dedicated multimedia data management Grid with no other computation intensive applications running simultaneously. In other scenarios, this basic load balancing algorithm should be extended to include the different real-time factors that would decide on the amount of data to be moved. Such modifications, specific to the Grid characteristics, should be possible without any loss of generality.

3.3.1 Basics of Load Balancing in a Distributed Environment

There are two approaches of dynamic load distributions: load-sharing and load-balancing. Where both load-sharing and load-balancing approach tends to maximize the rate at which distribution systems work, when required resources are available, load-balancing additionally attempts to equalize the loads on the available nodes [23]. Additionally, load distribution algorithms can be categorized as: Sender-Initiated Algorithms [10], Receiver-Initiated Algorithms [10], Symmetrically Initiated Algorithms [20] and Adaptive Algorithm [21]. As the name suggests, in Sender-Initiated Algorithms, load-distribution is initiated by an overloaded sender that tends to send a task to an under-loaded receiver. In the Receiver-Initiated Algorithms, load distributions is initiated from an under-loaded node (receiver) to a over-

loaded node (sender). For Symmetrically initiated algorithms, both the overloaded as well as the under-loaded nodes initiate the load-distribution and possess the advantages of both the Sender-Initiated as well as the Receiver-Initiated algorithms. The Adaptive algorithms attempts to address the issues that arise in the above three approaches. The main issue is the indiscriminate polling by the senders negotiation component. The adaptive algorithm maintains the state of the relationships between the sender and the receiver and adapts itself so as to scale well in larger systems. The load balancing algorithm proposed for our framework can be considered as the mixture of the sender-initiated and the adaptive approach. It considers the states of all the nodes of the distributed environment but essentially transfers load from the most loaded node to the least loaded. A mixed approach is utilized because Grid environments have an increasing potential to grow. Thus, any function developed for a Grid environment should be scalable. By keeping track of the states of the overall system, the different load balancing parameters (such as the amount of load that should be transfered, identification of the nodes whose loads should be balanced) can be adjusted.

4 Empirical Study

We carried out an extensive analysis of the performance of the different critical functionalities of the proposed framework with a varied data set and in a varied environment. As mentioned before, in this chapter we used images as the multimedia object type and all subsequent experiments were performed on them. We used about 9000 images from different categories, collected from the COREL dataset [9]. These 9000 images were distributed among the data repositories of the different nodes of the Grid. The simulated distributed/Grid environment has 8 Intel-based nodes. The total storage available for users is around 320GB. Each node is simulated by a Pentium 4 processor with Hyper Threading at 3GHz. The images are represented with 12 features comprising of colors and textures.

We divided the experiments into three categories. At first, we analyze the relationship of the computation cost with the number of distribution nodes while generating the index tree. The experimental results presented in Figure 3 demonstrates that as the number of distribution nodes increases, the average computation time (measured in seconds) decreases. The same data load is distributed over multiple nodes and they all run in parallel, thus decreasing the computation overhead of individual node. We performed k-NN search for about 15 queries and averaged the results. The computation time for each instance for each query is the maximum of the computation time among the distributed loads. This is because, the *Multimedia Application Interface* waits for the query results from all the nodes before providing the aggregate query result to the user. It is interesting to note that the computation overhead during the k-NN search has no direct relationship with the number of distribution nodes used. Each node handles the query individually and the time taken for completion it depends on the data set (both the data load as well as the data

content) in the particular node. As demonstrated in Figure 4, for Data Set A distributed over 4 nodes, the computation time increases steadily with the increase of the number of nodes. However, Figure 5 demonstrates that for a different data set B, the computation time drops when the number of nodes is 3 before rising again when number of nodes is 4.

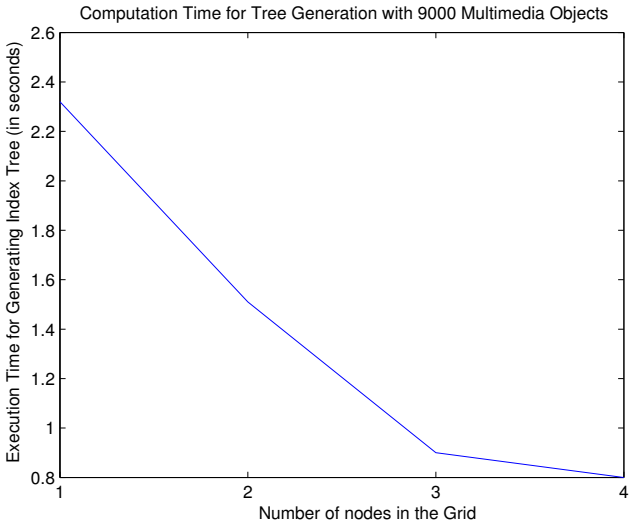

Fig. 3 Relationship of the Computation Time with the number of Distribution Nodes during Tree Generation.

The average accuracy of query results is about $80-85\%$. We deployed a distance-based index structure, M-Tree, which doesn't consider the high-level semantic relationships during the retrievals. The results obtained, although comparable in the computation overhead to the proposed framework, generated query results with very poor relevance (averaging as low as $15-20\%$).

The load balancing technique is demonstrated in Figure 6, 7, and 8, respectively. It should be noted that the load is balanced after different iterations for different data sets. We limited our examination for 5 iterations on an average, since in most of the cases we reached a considerable balanced load distribution within 5 iterations. Again, the variation is dependent on the data set used. In our experiment, we varied the number of data sets used in each case to bring a variation. Scenario I uses 500 data points, Scenario II uses 2300 data points and Scenario III uses 8500 data points, respectively.

From the detailed experimental analysis, it can be concluded that the proposed Distributed Multimedia Database Management Framework is capable of fulfilling the following requirements. First, it leverages the distributed environment of the Grid in economizing the computation overhead. Second, it is capable of supporting the popular multimedia retrieval requirements with relevant query results in a dis-

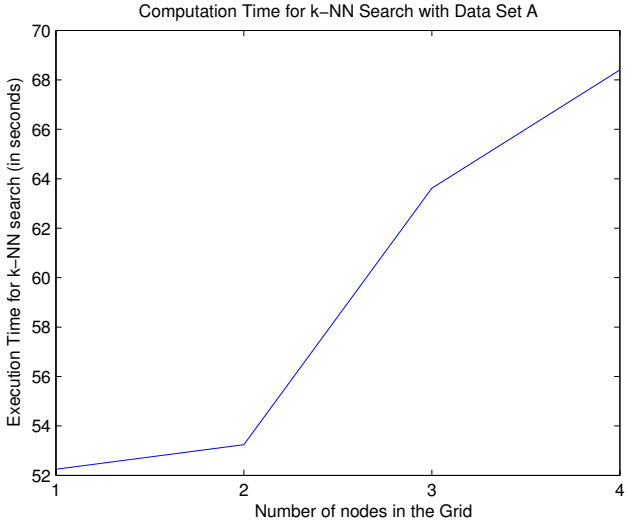

Fig. 4 Relationship of the Computation Time with the number of Distribution Nodes during k-NN Search for Data Set A.

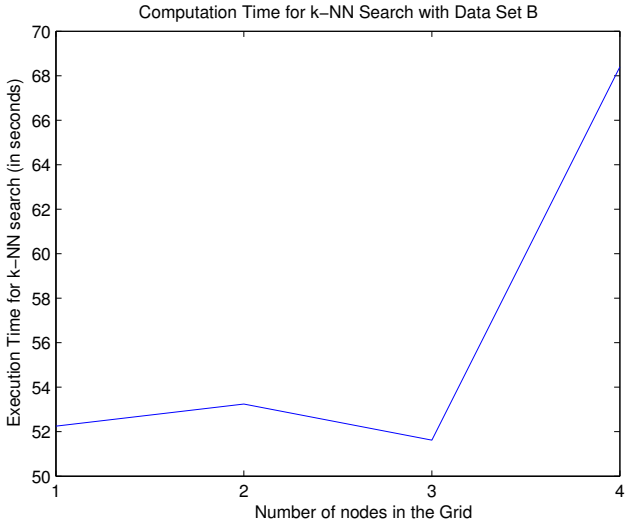

Fig. 5 Relationship of the Computation Time with the number of Distribution Nodes during k-NN Search for Data Set B.

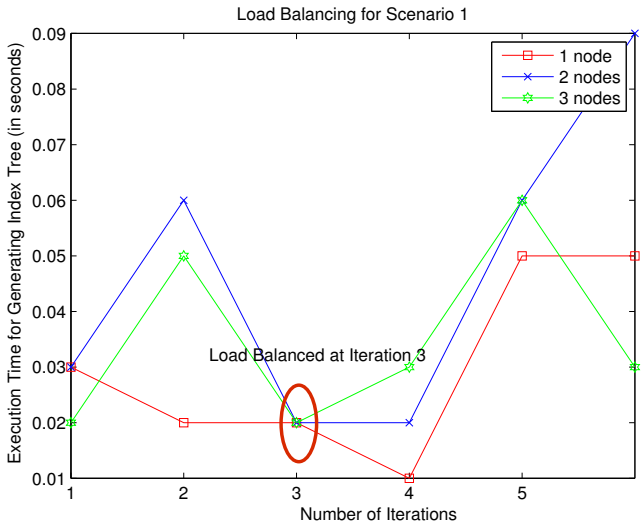

Fig. 6 Experimental Results for Load Balancing for Data Set I.

tributed environment. And finally, it successfully embeds functionalities typical to distributed environments, like a load balancing, into the multimedia environment, to make the proposed architecture adept for the Grid.

5 Conclusion and Future Works

In this chapter, we proposed a Distributed Multimedia Database Management Framework over a Grid. The framework introduced includes the important components necessary for storing and supporting Multimedia Applications over the Grid. A multidimensional replicated index structure was proposed that can support the popular multimedia retrievals based on contents. The framework introduces a stochastic construct, called the Markov Model Mediator, to capture and utilize the high-level semantic relationship among the multimedia objects. The novel inclusion of the high-level semantic relationship into the k-NN search algorithm, without violating the underlying indexed space, bridges the semantic gap and increases the relevance of query results manifold.

A load balancing approach for the multimedia data objects was also introduced, which successfully distributes the load across all the nodes of the Grid. In additions, intensive experimental analysis is performed with varied data set and different Grid configurations, which demonstrates that the proposed framework is a novel approach and a big step towards a full-fledged Multimedia Data Grid. The current framework can be extended in several directions. First, more Grid specific compo-

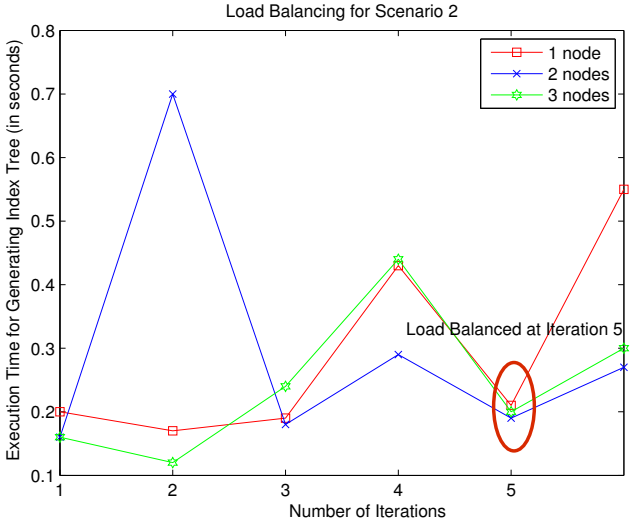

Fig. 7 Experimental Results for Load Balancing for Data Set II.

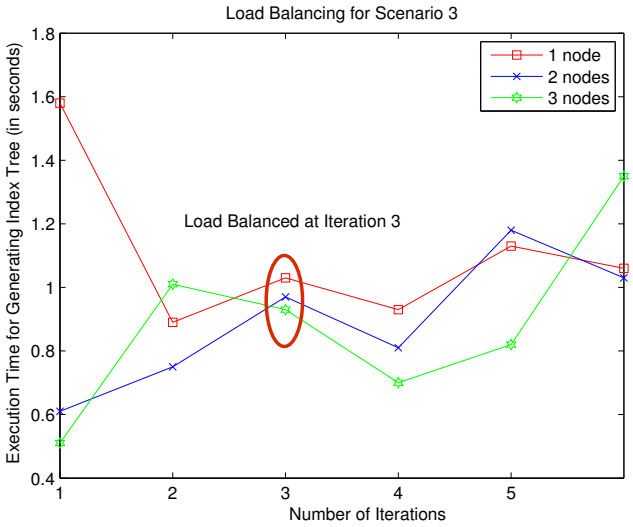

Fig. 8 Experimental Results for Load Balancing for Data Set III.

nents such as replica managers, auto failure detection and recovery of the Multimedia Data Nodes can be added. Second, the current framework should be extended to support other forms of multimedia data such as videos, and documents within one seamless platform. And third, developing Multimedia Grid Services such as

Content-Based Information Retrievals and Content-Based Multimedia search could be developed.

Acknowledgements This work was supported in part by the National science Foundation (grants OISE-0730065, OCI-0636031, and HRD-0833093) and in part by IBM. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the NSF and IBM. The authors would like to thank Mr. Dongri Luo for his help in enabling the Index structure to be executed in the Linux environment successfully.

References

1. R. Bayer (1971), Binary B-Trees for Virtual Memory in *Proceedings of SIGFIDET Workshop*, pp. 219–235
2. P. B. Berra, C. Y. R. Chen, A. Ghafoor, C. C. Lin, T. D. C. Little, and D. Shin (1990) Architecture for distributed multimedia database systems. Journal of Computer Communication, (13)4, pp. 217–231
3. K. Chakrabarti and S. Mehrotra (1999) The hybrid Tree: An Index Structure for High-Dimensional Feature Spaces in *Proceedings of the IEEE International Conference on Data Engineering*, pp. 440–447
4. K. Chatterjee and S.-C. Chen (2006) Affinity Hybrid Tree: An Indexing Technique for Content-Based Image Retrieval in Multimedia Databases in *Proceedings of the IEEE International Symposium on Multimedia (ISM06)*, pp. 47–54
5. K. Chatterjee and S.-C. Chen (2008) Hierarchical Affinity-Hybrid Tree: A Multidimensional Index Structure to Organize Videos and Support Content-Based Retrievals in *Proceedings of 2008 IEEE International Conference on Information Reuse and Integration*, pp. 435–440
6. K. Chatterjee and S.-C. Chen (2008) GeM-Tree: Towards a Generalized Multidimensional Index Structure Supporting Image and Video Retrieval in *Proceedings of the Fourth IEEE International Workshop on Multimedia Information Processing and Retrieval (MIPR2008), in conjunction with IEEE International Symposium on Multimedia (ISM2008)*, pp. 631–636
7. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke(2001) The data grid: towards an architecture for the distributed management and analysis os large scientific datasets. Journal of Network and COmputer Applications 23:187–200
8. P. Ciaccia, M. Patella, and P. Zezula (1997) M-tree: An Efficient Access Method for Similarity Search in Metric Spaces in *Proc. 23rd VLDB International Conference*, pp. 426–435
9. COREL STUDIO http://www.digitalriver.com/v2.0-img/operations/corelpps/desc/index.htm. Cited 29 March 2009
10. D. Eager, E. D. Lazowska, and J. Zahorjan (1986) Adaptive load sharing in homogeneous distributed systems. IEEE Trans. Softw. Eng., (12)5, pp. 662–675
11. G. Evangelidis, D. Lomet, and B. Salzberg (1995) The hB-Pi-Tree: A modified hB-tree supporting concurrency, recoverey, and node consolidation in *Proceedings of Very Large Databases Conference*, pp. 551–561
12. I. Foster, C. Kesselman (1999) The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers
13. I. Foster (2002) What is the Grid? - a three point checklist. GRIDtoday, (1)6
14. I. Foster (2003) The virtual data grid: a new model and architecture for data-intensive collaboration in *Proceedings of the 15th International Conference on Scientific and Statistical Database Management*, pp. 11–22
15. I. Foster, C. Kesselman, and S. Tuecke (2001) The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications, 15(3), pp. 200-222

16. A. Guttman (1984) R-trees: A Dynamic Index Structure for Spatial Searching in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47-57
17. L. Ivan, M. Ricarte, and C. M. Tobar (1996) Towards an Architecture for Distributed Multimedia Databases in *Proceedings of the 1996 IASTED/ISMM International Conference on Intelligent Information Management Systems*
18. T. Johnson and P. Krishna (1993) Lazy updates for distributed search structure in *Proceedings of ACM SIGMOD Conference*, pp. 337–346
19. B. Kroll and P. Windmayer (1994) Distributing a search tree among a growing number of processors in *Proceedings of ACM SIGMOD Conference*, pp. 265–276
20. P. Krueger and M. Livny (1987) The Diverse Objectives of Distributed Scheduling Policies in *Proceedings of the IEEE Symposium on Distributed Computing Systems*, pp. 242–249
21. P. Krueger and R. Chawla (1991) The Stealth distributed scheduler in *Proceedings of the 11th International Conference on Distributed Computing Systems*, pp. 336–343
22. H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman (2002) Data Replication Strategies in Grid Environments in *Proceedings of the 5th International Conference on Algorithms and Architecture for Parallel Processing*, pp. 378–383
23. M. Livny and M. Melman (1982) Load balancing in homogeneous broadcast distributed systems in *Proceedings of the Computer Network Performance Symposium*, pp. 47-55
24. D. Lomet (1990) Replicated Indexes for Distributed Data in *Proceedings of International Conference of Parallel and Distributed Information Systems*, pp. 1–8
25. M.-L. Shyu, and S.-C. Chen, and M. Chen, and C. Zhang, and C.-M. Shu (2003) MMM: A Stochastic Mechanism for Image Database Queries in *Proceedings 5th International Symposium on Multimedia Software Engineering (MSE2003)*, pp. 188–195
26. M. D. Stefano (2005) Distributed Data Management for Grid Computing. Wiley