PAMIR: A Parallel Multimedia Information Retrieval System

Adil Alpkocak, Tuba Ulker, Taner Danisman

Dokuz Eylul University
Department of Computer Engineering
35100 Bornova, Izmir, TURKEY
{alpkocak, tuba, taner}@cs.deu.edu.tr

Abstract. *This paper describes the design, implementation and evaluation of Parallel Multimedia Information Retrieval systems. PAMIR is implemented on a network of Pentium PC based workstations to exploit the I/O and computation parallelism in both insertion and processing phase of complex similarity queries in multimedia Information retrieval. The system is capable to index and query multimedia objects concurrently based on more than one features (i.e., global color, local color, texture, shape) and each features of an object queried by a different workstation using multithreaded version of M-tree.*

The system was tested for a collection of approximately 68000 images and each image described by three features; color moments, histogram layouts and texture. Experimentation has been shown that the system provides a good reduction in CPU times on both insertion and query processing phase. Detailed explanation of the system and experimentation results also presented.

1 Introduction

State-of-the art in content-based multimedia retrieval involves two main issues: f*eature extraction*, which extracts feature with distinguishing power from the original Multimedia Objects (MOB), and *feature indexing*, which organizes the objects in the database according to their extracted features in such a way that at run time, only a small portion of the database needs to be explored to retrieve the target objects. The features are extracted directly from the Multimedia Objects (MOB) with as less human intervention as possible. Feature extraction is a very old research area and the literature involves many of the well-known techniques to extract meaningful features from various types of signals such as images, video and audio. In content-based multimedia retrieval, the most common features for MOBs are shape, texture, color for still images, loudness and harmonicity for sounds, shots and objects' trajectories for video etc. Although these features are very common and successfully implemented by many content-based image retrieval systems, the feature extraction phase alone is still a CPU bound problem, particularly, when audio or video is the case.

In a Multimedia Information Retrieval System, the extracted features are then mapped into points in *d*-dimensional vector space and queries are processed against those features vectors. Therefore, a multimedia database can be envisioned as a collection of feature vectors representing MOBs. The Similarity of two images is defined as a distance of their representing feature vectors in *d*-dimensional vector space. In image databases, a typical similarity query corresponds to a nearest-neighbor or range query in *d*-dimensional feature space.

To date, many of the similarity indexing structures has been proposed such as TV-trees, R-trees, SS-trees, X-trees and M-tree. But, use of a similarity indexing is not generally fulfills the constraints about similarity searches. This is because, similarity indexing is both CPU and I/O bound due to the fact that the distance computations are CPU intensive tasks and image data can be too large. Moreover, contents of a MOB may have different properties, i.e., different dimensionality, and distance/similarity metrics. In other words, in a multimedia repository, each MOB is represented by a set of features vectors that may have different properties. Therefore, a multimedia retrieval system should be able to index every features used according to their properties.

On the other hand, content-based multimedia queries do not have to be based on one single feature, and the queries are generally based on the similarity of objects using several feature attributes like shape, texture, color or text. These types of multi-feature queries are called *complex similarity queries* consisting of more than one similarity predicate. Complex queries return a ranked result set with a calculated cumulative score. The results are presented to the user in rank order with highest scores indicating a better match. Within this context, a query based on only one feature is called as *atomic* query and a complex query is considered as a combination of atomic queries.

Some recent work has been done on processing complex similarity queries. In [CPZ97], the authors proposed a solution by extending M-tree indexing structure. However, they have concentrated on a situation where all similarity predicates refer to a single feature. In [CP00], another extension was proposed and original M-tree was renamed as M^2-tree, to perform complex similarity search over multimedia objects represented by multiple features. Both study focused on *feature indexing* issue of content-based multimedia retrieval.

In [Fag99], Fagin proposed a solution that the k best matches are returned for a complex query carried out by independent sub-systems, and that access to any one system is synchronized with the others. This algorithm is called A_0. It has problems of needing too many random accesses. A_0 algorithm formed a base for new researchers. In [NR99], another algorithm, based on Fagin's algorithm, was proposed and called a multi-step query processing algorithm. They claimed that it performs better than Fagin's algorithm by reducing number of database touches on random access phase. In [GBK00], the authors has modified Fagin's algorithm again, and called, quick-combine and proved that it was more efficient by a factor of 30. In [GBK01], authors presented a new algorithm called stream-combine, that allows retrieving the *k* top-scored objects from a set of different ranked result list without needing any random accesses where the order of the objects in resulting *k* set is not important.

Based on these observations, we present the parallel multimedia information retrieval (PAMIR) system, which is designed to address issues for both in *feature extraction* and *feature indexing*. PAMIR is implemented on a network of Pentium PC based workstations to exploit the I/O and computation parallelism in both search and insertion process of multimedia retrieval. The architecture of PAMIR involves a dedicated workstation for each features of MOB. Each workstation is responsible for both extracting respective features directly from MOB in insertion phase, and indexing those extracted features using modified multi-threaded version of M-tree. Although the main idea behind the PAMIR is quite simple it provides a good speed up in both database population and query processing response time, and it is scalable with the increasing number of features used in the system.

The remainder of the paper is organized as follows. Section 2 describes the problem, presents the architectural characteristics of PAMIR and explains the basic operations such as object insertion and query processing. In Section 3, we present the results obtained from the performance evaluation experimentations. Section 4 concludes the paper and provides an outlook for our future work on this subject.

2 Parallel Multimedia Information Retrieval Environment

A Multimedia Information retrieval system locates and retrieves objects that are relevant to user queries from database. A multimedia object can be an article, reports, image, chart, audio or video. So, multimedia information retrieval systems are designed with the objective of providing, in response to a user query, references to items that contain the information desired by the user.

Retrieving relevant multimedia objects from a large collection is of great importance in almost all types of multimedia information. Large databases create a performance problem and complex feature description consumes too much CPU time. While solving this problem, the throughput of the system must be increased and the response time of the individual queries must be decreased. In order to fulfill these requirements we believe that parallelism may help to this problem, therefore, we propose the PAMIR – Parallel Multimedia Information Retrieval system.

2.1 Basic Definitions

To clarify the later details, let us consider a multimedia system containing the set of multimedia objects, $MMS = \{M_1, M_2, ..., M_m\}$, where each multimedia object is defined by its feature vectors, $M_i = \{F_1, F_2, ..., F_n\}$,where $1 ? i ? m$, and a feature vector is defined by its properties $F_i = < f_{i1}, ..., f_{1p}>$, where $1 ? i ? n$ and each F_i may have different dimension. In addition, let a set of similarity (distance) functions, $SIM=\{ SIM_1, ..., SIM_n\}$, be defined to calculate spatial proximity between *F*s, where *SIM*s may not be identical.

A multimedia query *Q* can be formulated using these feature vectors as follows:

$$Q = \{F_1, F_2, ..., F_n\}$$

In a system like this, the query processor requires an organization, which facilities locating relevant objects with respect to their distinct features.

The main issue for parallel system is how we distribute all the multimedia information in the whole collection to the processors in such a way so that all advantages of parallelism could be handled. So, the problem is how to partition the data? There may be two methods for partitioning the data over the processors. These are *horizontal partitioning* and *vertical partitioning*.

To clarify how portioning handled, let us consider an example involving an image database where each image is described by three feature vectors: color histogram, texture and shape information of the image. More formally, the image database has a set of images, i.e., multimedia objects,

$$MMS = \{I_1, I_2, ..., I_m\}$$

where I_i is an image object and each image object is described by three feature vectors, $I_i = \{C_i, T_i, S_i\}$, where C_i is color histogram vector, T_i is texture information vector, S_i is shape information vector of image object i. Each feature vector for image I_i can be defined as follows:

$$C_i = < c_{i1}, ..., c_{in} >$$
$$T_i = < t_{i1}, ..., t_{ip} >$$
$$S_i = < s_{i1}, ..., s_{ir} >$$

where n, p and r are the number of entries in the feature vectors. So, an image can be represented as follows:

$$I_i = \{ < c_{i1}, ..., c_{in} > , < t_{i1}, ..., t_{ip} > , < s_{i1}, ..., s_{ir} > \}$$

Alternatively, we can show the image database with their corresponding features as a matrix as follows:

$$MMS = \begin{bmatrix} C_1 & T_1 & S_1 \\ C_2 & T_2 & V_2 \\ ... & ... & ... \\ C_m & T_m & S_m \end{bmatrix}$$

Partitioning can be done horizontally (row-wise) or vertically (column-wise). In this study, we focus on vertical portioning due to the main issues of multimedia information retrieval (e.g., feature extraction and feature indexing). We propose vertical partitioning of data over the processors so that different feature algorithms and different indexing schemes can be applied to each feature.

2.2 Architecture

Figure 1 shows a typical architecture of PAMIR with one master and three slaves: one for color, one for texture and one for shape. Color vectors of all objects are stored in

first slave, texture vectors of all images are in second slave and so on. A typical query can be processed as follows. The master processor gets the query by interacting by user and sends the request to slaves. Slaves process the query request concurrently and then master collects the each slave's response. Finally, it merges and outputs the results to user.

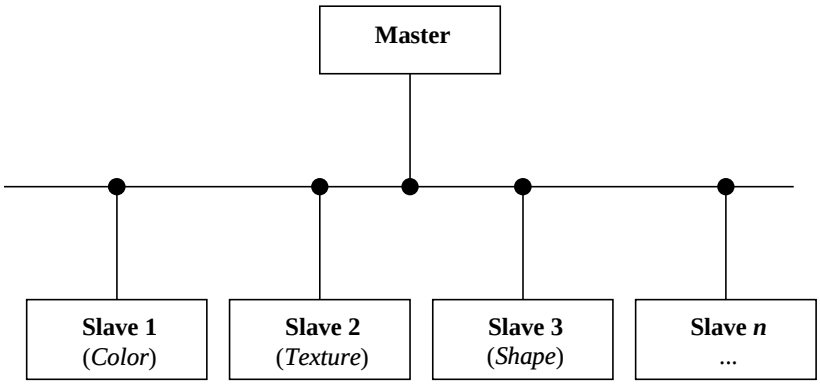

Figure 1. PAMIR architecture

When a new object is to be inserted into the database, firstly it is necessary to extract the feature vectors of object. To do this, object is broadcasted to all slaves. Then, each processor extracts its respective feature vector applying proper algorithm and stores the vector in its own disk. Each processor can use different indexing methodology.

2.3 Query Processing

A Multimedia Information Retrieval Systems must be enable to find all objects in the region which is defined with a specific distance from a given query object in large linearly ordered domains of attribute values. For an image database, a typical query can be posed in natural language as, "*find all images having the red circle*". In this example, the query has two predicates: one for color feature (red) and another one for shape feature (circle). However, these feature vectors describing different media contents have different properties, i.e., different dimensionality and similarity or distance measure metrics.

If a multimedia query is formed with more than one predicate, as in above example, each of which corresponds a single feature then such queries are called *Complex queries*. Complex queries may have a different similarity function from single feature queries. This is because the user wants to retrieve the best matches to the whole query. Complex queries must be a combined function that compares all features available in the query. Fagin's A_0 algorithm provides a solution for multiple features at the same time. Based on Fagin's algorithm, some other researchers offered some improvements [NR99][GBK00]. We will explain these three algorithms and their parallel versions in the following sections.

Fagin's A_0 Algorithm

For a given complex query, (Color = "Red") $\wedge$ (Shape = "Round"), the solution set of this query is a "graded" (or "fuzzy") set. A graded set is a set that is given by some pairs such as (x, g), where x is an object, and g is a real number in the interval [0,1]. It can also be considered that a graded set is a sorted list, where the objects are sorted by their grades.

For each sub-queries each handles a single feature predicate, a grade is assigned to each object. The grade represents the degree of fulfillment of object on this sub-query, where the larger the grade is, the better the match. In particular, a grade of 1 represents a perfect match. For traditional database queries, the grade for each object is either 0 or 1, where 0 means that the query is false about the object, and 1 means that the query is true about the object. For multimedia queries, grades may be intermediate values between 0 and 1. Boolean combinations of sub-queries are processed by some "aggregation functions" that assign a grade to a fuzzy conjunction, as a function of the grades assigned to the conjuncts.

If x is an object and Q is a query, let $m_Q(x)$ the grade of x against the query Q. Based on definitions and boolean combinations of sub-queries are defined by the standard fuzzy logic given below;

Conjunction rule : $m_{A \grave{U} B}(x) = \min \{m_A(x), m_B(x)\}$
Disjunction rule: $m_{A \acute{U} B}(x) = \max \{m_A(x), m_B(x)\}$
Negation rule: $m\emptyset_A(x) = 1 - m_A(x)$

These are the 2-ary aggregation functions because there are 2 queries (A and B) are combined. What else if more than 2 queries? Such these queries are called m-ary queries. Let $F(A_1, \ldots, A_m)$ be an m-ary query, where $A_1, \ldots, A_m$ are query terms. The grade of each object under m-ary query is defined as $m_{F(A1, \ldots, Am)}$. The standard fuzzy logic semantics of the conjunction $A_1 \wedge \ldots \wedge A_m$ is given by defining

$$m_{A1 \grave{U} \ldots \grave{U} Am}(x) = \min \{m_{A1}(x), \ldots, m_{Am}(x)\} ,$$

for each object x. Let . Let t be an m-ary aggregation function.The m-ary query $F_t(A_1, \ldots, A_m)$ is defined by

$$m_{Ft(A1,\ldots, Am)}(x) = t\,(m_{A1}(x), \ldots, m_{Am}(x)) .$$

Fagin's A_0 algorithm was used in Garlic project, for combining multiple features. Both conjunctive and disjunctive queries can be processed using A_0 algorithm. The algorithm returns the top k answers for a query with multiple features $F(A_1, \ldots, A_m)$, which is denoted by Q. It consists of three phases: sorted access, random access and computation phase.

1. There is a subsystem i that evaluates the sub-query A_i. For each i the corresponding subsystem outputs a set one by one in sorted order based on grade, the graded set consisting of all pairs $(x, m_{Ai}(x))$, where as before x is an object and $m_{Ai}(x)$ is the grade of x under query A_i.

This process is repeated until the intersection of the m lists is of size at least k. That is, repeat until there is a set L of at least k objects such that each subsystem has output all of the members of L.

2. For each object x that has been seen, do a random access to each subsystem j to find $m_{Aj}(x)$. If x is in the graded set of sub-query A_j, there is no need to do a random access for x in subsystem j.

3. Compute the grade $m_Q(x) = t(m_{A1}(x), \ldots, m_{Am}(x))$ for each object x that has been seen. Sort the objects on their grades and form a result set. The output is then selected from this set with teh top k objects.

Multi-Step Query Processing Algorithm

This algorithm was proposed in [NR99] and claimed that it performs better than Fagin's algorithm, because it requires fewer database accesses. It was shown that, to improve Fagin's algorithm, combining function must satisfy two additional properties defined below.

Definiton 1 *Lower bounding property*: A combining function t for the objects x and x' under the query $A_1 \wedge A_2 \wedge \ldots \wedge A_m$ satisfies the lower bounding property iff

$$\exists i \forall j : m_{Ai}(x) \; ? \; m_{Aj}(x') \; ? \quad t(m_{A1}(x), \ldots, m_{Am}(x)) \; ? \; t(m_{A1}(x'), \ldots, m_{Am}(x'))$$

Definiton 2 *Upper bounding property*: A combining function s for the objects x and x' under the query $A_1 \vee A_2 \vee \ldots \vee A_m$ satisfies the upper bounding property iff

$$\exists i \forall j : m_{Ai}(x) \; ? \; m_{Aj}(x') \; ? \quad s(m_{A1}(x), \ldots, m_{Am}(x)) \; ? \; s(m_{A1}(x'), \ldots, m_{Am}(x'))$$

Again, the standard rules of fuzzy logic for conjunction, disjunction and negation are used from combining functions. The standard fuzzy rule "min" satisfies the lower bounding property while the standard fuzzy rule "max" satisfies the upper bounding property.

Multi-step query processing algorithm returns the top k objects for a query $Q=F_t(A_1,\ldots, A_m)$. Each subsystem i, $1 \; ? \; i \; ? \; m$, returns the matching objects in sorted order. The termination condition is to fill the result list with at least k objects. This process is an iterative process.

1. For each query term A_i, request the subsystem i to return the next object x. Thus, each subsystem i outputs the graded set of pairs $(x, m_{Ai}(x))$, where x is an object in the database and $m_{Ai}(x)$ is the similarity value of x under A_i.

2. For each object x returned by the subsystem i in the current iteration, do random access to subsystems j, $i \neq j$, to find $m_{Aj}(x)$.

3. Compute the threshold grade, t_h for this iteration, $t_h = t(m_{Ai}(x_i), \ldots, m_{Am}(x_m))$ where x_i is the element retrieved by subsystem i in the current iteration.

4. Compute the grade $m_Q(x) = t(m_{Ai}(x), \ldots, m_{Am}(x))$ for each object x retrieved in this iteration. Update Y, the set containing all objects that have grade $m_Q(x)$? t_h.

5. Go to next iteration (repeat steps 2 to 5) until the set Y has k objects.

6. Output the graded set $\{(x, m_Q(x)) | x \in Y\}$

Their theorem on this algorithm is "*For every query $F_t(A_1, \ldots, A_m)$, the multi-step algorithm correctly returns the top k answers, if t satisfies the bounding properties*". From their observations, Fagin's algorithm requires more database accesses. Because, when the lower and upper bound properties are satisfied by the combining function used, the number of random accesses for non-available grade values can be reduced.

Quick-Combine Algorithm

Quick-Combine algorithm proposed in [GBK00] is similar to multi-step algorithm in the property of iterative process. But in this case, instead of computing a threshold value, present-top scored object is found in each step. Termination condition is to retrieve a list that has at least k objects.

2.4 Combining Complex Queries in PAMIR

In PAMIR, complex queries are processed based on the algorithms mentioned before. However, we have parallelized all tree algorithms and the details are given in Algorithm 1, 2 and 3.

Algorithm 1. Parallel version of Fagin's A_0 Algorithm

Step 1. Master gets the user query in the form of boolean combination of sub-queries, i.e. Q = (Color="Red" and Texture = "Fine" and Shape = "Circular")

Step 2. Master divides the whole query into sub queries according to the predicates of each features, i.e. Q_1 = (Color = "Red"), Q_2=(Texture="Fine"), Q_3=(Shape="Circular")

Step 3. Master sends each sub-query to the corresponding slaves and requires the sorted lists from each slave, i.e. 3 sorted lists each of which has k elements for color, texture and shape are required. This step meets the sorted access phase of Fagin's algorithm.

Step 4. Each slave evaluates its own sub-query and sends the result sets back to master.

Step 5. Master merges all lists and forms a single set. The objects in this set may not have similarity values for all features. That is, there may be non-available grade values for object. Master determines the non-available grade values of each object and send a request for grade value to the corresponding slave processor, i.e. non-available grade values for color feature are sent to slave 1, non-available grade values for texture feature are sent to slave 2, non-available grade values for shape feature are sent to slave 3. This step meets the random access phase of Fagin's algorithm.

Step 6. Each slave gets its request and begins to compute the similarity values for objects. Then, they send the results back to master.

Step 7. Master computes the similarity values of each object under the overall query (Q) and sorts them on their similarities. Finally, it outputs the top k objects (highest k objects) to user. Last step meets the computation phase of Fagin's algorithm.

Algorithm 2. Parallel version of Multi-Step Query Processing Algorithm

Step 1. Master gets the user query in the form of boolean combination of sub queries, i.e. Q = (Color = "Red" and Texture = "Fine" and Shape = "Circular")

Step 2. Master divides the whole query into sub queries according to the predicates of each features, i.e. Q_1 = (Color = "Red"), Q_2 = (Texture = "Fine"), Q_3 = (Shape = "Circular")

Step 3. Master sends each sub-query to the corresponding slaves and requires the sorted lists from each slave, i.e. 3 sorted lists each of which has k elements for color, texture and shape are required.

Step 4. Each slave evaluates its own sub-query and sends the result sets back to master.

Step 5. Master doesn't merge all lists. It picks just one object from returned list. And sends this object to all slaves except the subsystem that the object belongs it to compute the missing grade values. This phase includes random access for one object.

Step 6. Master computes a threshold value based on all subsystems. Then for selected object the grade value of overall query is computed from the grade value under all subsystems. If the grade of this object is bigger than threshold value then add it to the final list.

Step 7. Go to step 5 and stop until the final list has at least k objects.

Step 8. Finally, show the graded set of k objects to the user.

Algorithm 3. Parallel version of Quick-Combine Algorithm

Step 1. Master gets the user query in the form of boolean combination of sub queries, i.e. Q = (Color = "Red" and Texture = "Fine" and Shape = "Circular")

Step 2. Master divides the whole query into sub queries according to the predicates of each features, i.e. Q_1 = (Color = "Red"), Q_2 = (Texture = "Fine"), Q_3 = (Shape = "Circular")

Step 3. Master sends each sub-query to the corresponding slaves and requires the sorted lists from each slave, i.e. 3 sorted lists each of which has k elements for color, texture and shape are required.

Step 4. Each slave evaluates its own sub-query and sends the result sets back to master.

Step 5. For returned objects, master picks p objects among them where p is a suitable natural number. It sends these objects to all slaves except the subsystem that the object belongs it to compute the missing score for every sub query by random access. For each new object there are $(n - 1)$ random accesses necessary.

Step 6. Master checks if the k top-scored objects are the best k objects of the database. It compares the aggregated score of the present k top-scored objects to the value of the combining function with the lowest scores seen from each feature. Then, it checks if there are at least k objects whose aggregated score is larger or equal than the aggregated minimum scores per feature.

Step 7. Master checks if the k top-scores of k objects are bigger than the minimum scores then the k top-scored objects can be returned as top objects of the whole database. I this case, add them into the final list and go to step 8. If not, more elements of the atomic output streams have to be evaluated. In this case, go to step 5 for next p objects and stop until the final list has at least k objects.

Step 8. Finally, show the graded set of k objects to the user.

3 Performance Evaluation

The current implementation of PAMIR is built on top of four Pentium MMX based workstations (one master and three slaves), running at 233 Mhz processor speed. Master has 128 MB memory, and the rest have 64 MB. The workstations are linked via a 100 Mbit/s Fast Ethernet under the operating system of is Linux. The communication among the processors provided using MPI (message passing interface).

The system is tested on a collection of 68040 images from various categories. This dataset contains image from a Corel image collection. Images are described with three features based on the color histogram, color moments, and co-occurrence texture.

For color histogram, HSV color space is divided into 32 subspaces (32 colors: 8 ranges of H and 4 ranges of S). In the resulting histogram, the value in each dimension in a color histogram of an image is the density of each color in the entire image. Histogram intersection was used to measure the similarity between two images.

Color moments, we obtained 9 dimensional vectors (3 x 3) (one for each of H,S, and V in HSV color space) mean, standard deviation, and skewness. Euclidean distance between color moments of two images is used to represent the dis-similarity (distance) between two images.

The last feature used in our experimentation is for texture. Texture features are extracted by co-occurrence matrix, resulting 16 dimensions (4 x 4). Images are first converted to 16 gray-scale images and co-occurrence matrix in 4 directions is computed (horizontal, vertical, and two diagonal directions). The 16 values are (one for each direction) second Angular Moment, Contrast, Inverse Difference Moment, and Entropy. Euclidean distance between co-occurrence matrixes of two images is used to measure the dis-similarity (distance) between two images.

In order to evaluate the system, we have done two experimentations. First, we tested different combining algorithms for complex queries on our parallel environment for varying k values for a 100 successive queries. Figure-2 shows the results we obtained from our experimentation. As k value increases Multi-step and Quick-combine algorithms are seemed to be approaching to Fagin's algorithm due to increased communication cost among processors.

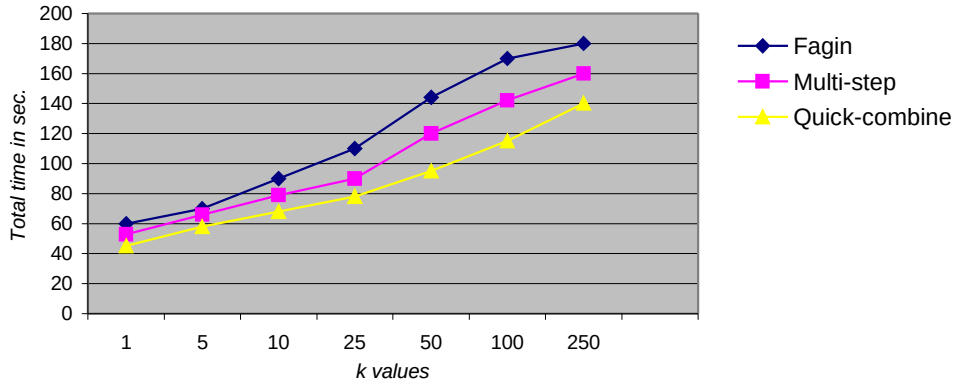

Figure 2. Result of combining algorithms comparison experiment.

The second experimentation we did is about scalability of PAMIR with increasing number of features. The results are shown in Figure 3. We have added one more features in each step and observed the total time for 100 successive queries again. As we added new features to PAMIR the changes in total time caused by increased communication cost. So, we can say that the PAMIR is scalable with increasing number of features.

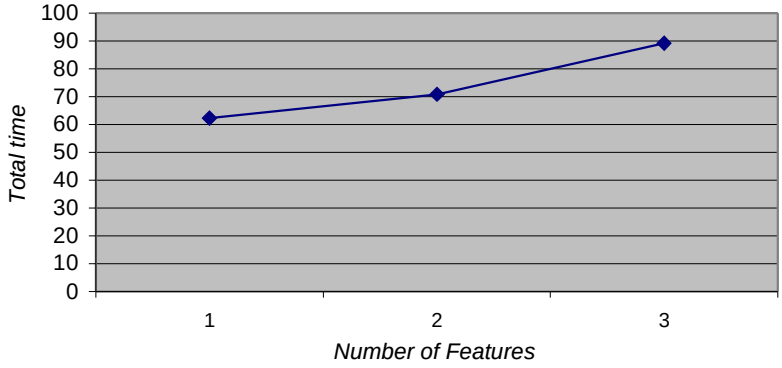

Figure 3. Result of Scalability experimentation

4 Conclusion

This paper presents the design, implementation, and evaluation of a parallel multimedia information retrieval system called PAMIR to exploit the I/O and computation parallelisms specifically for multi-feature complex query. We have successfully implemented the first PAMIR prototype on a network of Pentium workstations, and carried out a comprehensive performance evaluation of the

prototype against a database consisting of 68040 images where each image has been described by three features, color histogram, color moments and co-occurence texture.

In complex query processing phase, we have evaluated Fagin's, multi-step and quick-combine in our parallel environment. The performance evaluation experiments showed that quick-combine algorithm outperforms the others. Stream-combine algorithm is also a good solution if the order in final result set is not important. We have presented both all experimentation results and parallel version of respective algorithms for PAMIR environment.

PAMIR is a prototype system for parallel Multimedia information Retrieval, which is designed to address issues for both in *feature extraction* and *feature indexing* phases and it helps to process complex queries. PAMIR is implemented on a network of Pentium PC based workstations to exploit the I/O and computation parallelism in both search and insertion process of multimedia retrieval. PAMIR provides a good speed up in both database population and query processing response time, and it is scalable with the increasing number of features used in the system. In longer term, we expect the PAMIR project to lead us into new research in many dimensions, including query processing technologies, user interfaces, parallel similarity indexing and more.

References

[CH+95] W.F.Cody,L.M.Haas,W.Niblack, M.Arya,M.J.Carey, R. Fagin, M. Flickner, D. Lee, D. Petkovic, P. M. Schwarz,J. Thomas, M. T. Roth, J. H. Williams, and E. L. Wim-mers. "Querying multimedia data from multiple repositories by content: the Garlic project", *Proceedings of Third Working Conference on Visual Database Systems, VDB-3*, Lausanne, Switzerland, March 1995.

[CP00] Paolo Ciaccia, Marco Patella, "The M2-tree: Processing Complex Multi-feature Queries with just One Index", First Delos Workshop on Information Seeking, Searching and Querying in Digital Libraries, Zurich, Swithzerland, 2000.

[CPZ97] Paolo Ciaccia, Marco Patella, Pavel Zezula, "M-tree: An efficient access method for similarity search in metric spaces", *Proceedings of the 23rd VLDB International Conference*, Athens, Greece, September 1997.

[CPZ99] Paolo Ciaccia, Marco Patella, Pavel Zezula, "Processing Complex Similarity Queries with Distance-based Access Methods", *Proceedings of the 6th EDBT International Conference*, Valencia, Spain, March, 1998.

[Fag99] Ronald Fagin, "Combining fuzzy information from multiple systems", *Journal of Computer and System Sciences*, Vol. 58, pp. 83-99, 1999.

[GBK00] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling, "Optimizing Multi-Feature Queries for Image Databases", In *Proceedings of the 26th*

International Conference on Very Large Databases (VLDB 2000), pp. 419-428, Cairo, Egypt, 2000.

[GBK01] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling, "Toward Efficient Multi-feature Queries in Heterogeneous Environments", Proc. Of the IEEE International Conference on Information Technology: Coding and Computing (ITCC 2001), Las Vegas, USA, 2001.

[NR99] Surya Nepal and M.V. Ramakrishna, "Query Processing Issues in Image(multimedia) Databases", *Proceedings of the 15th International Conference on Data Engineering* , Valencia, Spain, March, 1999.

[WH+99] Edward L. Wimmers, Laura M. Haas, Mary Tork Roth, Cristoph Braendli, "Using Fagin's Algorithm for Merging Ranked Results in Multimedia Middleware", *Proceddings of 4th IFCIS International Conference on Cooperative Information Systems*, Edinburgh, SCOTLAND, September 1999.