CHAPTER 6

DISTRIBUTED ALGORITHMS FOR IMAGE DATA BASE CLASSIFICATION AND RETRIEVAL

6.1 INTRODUCTION

In the previous chapter, retrieval based on color and texture-based features were discussed by considering them separately and in combination. The performance of the retrieval system depends on the dataset used and the content-based features which are used for retrieval. When the time and space requirements of the features increase, the accuracy of retrieval is improved. As the database size increases, the sequential mode of retrieval takes more time. To reduce the retrieval time, a concurrent mode of searching operation is attempted in this thesis, to retrieve the images using the content of the images in a set of networked computers. The details of such network based retrieval are discussed in this chapter.

The need for computing resources is constantly increasing. This has been the driving force behind the rapid improvement of computer hardware that is now closer than ever to the limits set by the laws of physics. Distributed computing has been developed to improve the performance of computer systems by distributing the computations, and this has lately attracted interest, since it promises high-performance computing at a lower cost than traditional high-end computer systems. Grid computing attempts to provide a software application framework to computing services.
This research work investigates the use of distributed computing for task distribution. We implement a prototype in Windows XP to provide a single point-of-entry to a cluster of 4 computers that run a problem solving application. Requests are distributed on the cluster using the Message Passing Interface.

The development of new applications and increasing need for information processing has led to a constant lack of computing resources. This has been one of the driving forces for the computer and semi-conductor industries and so far the rate of growth of chip and computer speed has kept a steady pace. The speed increase is, however, accompanied by some problems. Even though circuits are built with technology, that is many times more efficient that of older models, the increased performance has led to higher power consumption, increased heat and power loss, and a constant struggle with the laws of physics. Computers with the latest technology, have very good performance capabilities when it comes to processor, memory and I/O devices, but another major development has been in the area of networking technologies. Fast network communication has become affordable and widespread.

Distributed computing is the area of computer science that deals with the development of applications that run parallel on multiple processors, or computers taking advantage of the increased computational power of each processor. The access to high performance computers and networking hardware has opened up possibilities for the extensive use of distributed computing in many areas. The roots of distributed computing lie in the scientific community, where high performance was often required at any cost, forcing the development of new ideas; but distributed computing technology is increasingly interesting and has become accepted in other business areas, where it promises cheap, high-performance and fault-tolerant computer systems.
This project intends to investigate and study distributed computing mainly in the areas of task distribution for CBIR.

6.2 TASK DISTRIBUTION

According to www.dictionary.com a “task” can be defined as “A piece of work assigned or done as part of one's duties.” In a distributed computer system, a task can be just about anything such as running a database query, analyzing some data or even starting an application. The type of tasks the system should be able to handle, must be defined when the system is designed.

The object of “task distribution” in a distributed system is to assign each task to one (or more) of the nodes in the system that is to execute the task. This should be done (if possible) following an algorithm that enables each task to be executed properly according to its requirements, while respecting the system constraints. A task requirement, for example, is that a task is performed within a certain time limit. System constraints can, for example, be the number of processors in the system and their speed, which limits how fast tasks can be executed. Task distribution should try to optimize some function, for example, minimizing the total execution time or maximizing the resource usage. This project considers task distribution in the context of distributed computing. We intend to investigate and construct a system that performs task distribution. The task distribution algorithm used in such a system is very important since it is responsible for using properly the resources available.
6.3 DISTRIBUTED COMPUTING

Distributed computing is a term often used to describe programs or computations that run on several processors or computers simultaneously. Often the idea is that some sort of cooperation should take place between the processors or computers. To make this possible, there has to be some kind of a communication system, often something like a memory. Lately, the term “Grid computing” has spread and it has gained wide acceptance as a term representing a concept.

Throughout this report, we may alternate between the terms “cluster”, “grid” and “grid farm”, whose meaning is “a set of interconnected computers that are used to solve problems cooperatively”. A “cluster” is often referred to as a set of computers specifically installed for the purpose of performing computations, often with a dedicated network connecting them. The term “grid” is associated with grid computing and can be seen as a bit more focused on the application framework that the system is installed on. Thus, a “grid farm” is also often used in the context of grid computing, but in this report, we consider a grid farm to be the hardware part of the system, the same as with the term ‘cluster’.

6.4 PROGRAMMING FOR DISTRIBUTION

Initializing and using a set of networked computers for running distributed applications is attractive, especially for some applications. One reason is that it might be possible to use existing systems, which would provide good performance at a low cost. Applications that use features such as threads or communicating processes might not at all be suitable for distribution. This is because they are often written with smaller granularity and the increased latency and decreased bandwidth will cause performance
problems. When designing and implementing a distributed application, these factors must be considered.

A computer program consists of a set of instructions that perform different actions depending on the data set it is executed with. The main problem in a distributed system is that, access to data, both static such as a file, and dynamic such as interaction with other processes or computers, is sometimes time-consuming. Reading and writing data of this kind must be minimized in order to get decent application performance.

A programming paradigm suitable for a distributed application should provide an interface that enables the developer to write the correct and efficient code. By forcing the programmer to access “expensive” resources as little as possible application performance can be maintained. We describe below two widely used paradigms for distributed programming.

6.4.1 Shared memory

A multiprocessor computer is a computer that has a number of processors (or computation elements) all of which the same memory usually through (at least logically) the same memory bus. In this system, a memory read or write is usually fast since it is done over the fast memory bus. If two processors, or processes running on separate processors, wish to communicate with each other, they can do this by simply writing to and reading from some defined memory regions. Figure 6.1 shows simple shared memory multiprocessor with three processors.
There are some issues that need to be addressed in a shared memory system. For example, some logic has to make sure that two processors do not write to the same memory at the same time, since this would produce data inconsistency errors. Modern systems also include a local cache inside each processor. It forces the use of cache consistency logic to make sure that the local caches are consistent at all times. A multiprocessor system can, however, handle this locking of memory efficiently because the system is capable of high-speed and low-latency communication through the memory bus.

A shared memory programming paradigm can also be implemented in distributed memory architecture (Tanenbaum and Steen 2003) but since it is normally used in an environment where latency and bandwidth is hidden, its use might cause performance problems in a system with higher latency and lower bandwidth.

6.4.2  Message passing

Message passing is a paradigm that is often used in machines with distributed memory architecture. The inter-process communication is based on passing messages between the processes, and the messages are transferred either locally through a memory bus in a multiprocessor system, or across a network in a distributed system. Because each message comes with a time-
penalty, limiting the number and size of the messages is likely to improve application performance. This will allow the developer to intuitively understand application bottlenecks and circumvent them.

In its basic form, message passing requires each message to have a sender, a receiver and some data. There is a small delay between the sender, sending a message and a receiver receiving it that depends on the latency of the communication system and the message size in combination with the bandwidth. It may also be possible to convey special messages such as broadcasts. A message from one sender may also be sent into many users.

6.4.3 Issues and potential

While designing and implementing a distributed application a major issue is that in order to get an efficient result, the design has to solve issues related to large latency, limited bandwidth, asynchronous execution between the nodes and the possibility of node failure. It is simply impossible to produce efficient distributed versions of often because of the nature of the algorithm or application. In other cases, distributed versions that have good performance can be produced very easily.

There is always some overhead associated with the distribution of an algorithm. The aim of producing a good distribution is to minimize this overhead. The ideal case would be that each node addition to the system would increase its performance by the individual performance of one node. This is rarely possible, but still a smaller performance increase will produce a system with better performance, and in some cases it might be the only or at least the simplest way of increasing application performance.
6.4.4 Feasible problems

A large number of scientific problems can and have been proven to benefit from the use of distributed computer systems. Normally these problems have some special characteristics that make them suitable for running in a multiprocessor system.

It has been proved possible to develop distributed versions of a lot of mathematical operations, e.g., matrix operations. It is even possible to break up the matrices in strips and distribute them to the nodes that perform the calculations required on each strip. The overhead comes from the breakup of the matrices and the merging of the results for the final answer. The performance will be affected by size of the problem and the number of nodes involved.

6.5 PROGRAMMING TOOLKITS

The development of a distributed application is often difficult. Not only must the algorithm be constructed in a way that enables it to be distributed, but other practical issues such as how to start the execution, distribute processes, keep track of the execution, handle data transfers and share resources must also be solved. A programming toolkit is a major aid in this area.

A programming toolkit provides a set of tools that offer ready-made implementations of certain functions. A distributed toolkit should provide tools for functions that simplify the development of a distributed application, and it should also take advantage of the underlying hardware. The exact functionality that is necessary, is based on what the toolkit is to be used for. Various toolkits with different functionalities have been developed, such as a
number of different projects at IBM T. J. Watson Research Center, Intel's NX/2, Express, nCUBE's Vertex, p4 and PARMACS, Zipcode, Chimp, PVM (Geist Al et al 1994), Chameleon, PICL and MOSIX (http://www.mosix.org/).

A toolkit is usually developed to satisfy a specific requirement either from applications or from a system hardware. Each toolkit will thus usually have some stronger and weaker points, and the choice of toolkit will depend on these properties. For a long time, there was no standardized toolkit, but the interest in distributed computing with cheap computer hardware spurred the development of the free PVM toolkit described in the following subsection. Having a standardized toolkit would help both hardware and software developers, and eventually, the MPI is described as a standard after a joint standardization process between many parties as discussed subsequently.

Toolkits such as PVM and MPI have made life easier for developers that want to write distributed applications. However, the toolkits still operate on a rather low level. The vision of Grid computing expands further, when it comes to functionality by adding features for security, data access, resource allocation, monitoring and service discovery.

6.5.1 Parallel Virtual Machine

The Parallel Virtual Machine (PVM) (Geist Al et al 1994) project began in 1989 at the Oak Ridge National Laboratory. The toolkit was based on message passing and was usable both in multiprocessor machines as well as in distributed environments or a mixture of both. The key concept in the PVM was that it made a collection of computers appear as one large virtual machine, hence the name.
Application development in the PVM was done using the concept of communicating processes. Logically, the developer started a number of processes that could communicate with each other through an interface. Whether the processes were executed locally or on a remote host was decided at run-time.

The PVM was designed to be versatile and it supported both data parallel programming and function parallel programming. Data parallel programming is when the data set of the problem is split up between the different nodes, but each of them executes the same basic logic. In function parallel programming, different nodes are responsible for different functions. The possibility of using data parallel or function parallel programming, or even a mixture of both, made the PVM useful in a different environment and was a key to its success.

The PVM was completely free and quickly became the de-facto standard for developing parallel applications. The main competitors of the time were toolkits developed by hardware manufacturers, but unlike the PVM applications, made with these were hardly portable. The PVM provided the basic functionality needed by parallel applications such as a message passing interface, synchronization and the possibility to start and stop applications across the network.

Interestingly, one feature which the PVM included, was the possibility to dynamically resize the virtual machine by allowing nodes to join and leave the computation at run-time. This feature also allowed for the construction of fault-tolerant applications. However, it also adds complexity since an application must handle nodes that join or leave.
The PVM has been very successful and popular because developers liked its features and interface. A lot of the functionality was borrowed when designing The MPI (which we shall describe in the next section). The PVM never became standardized and thus never became as widespread as it perhaps could have become, but many of its features were borrowed when MPI was constructed. We shall refer to (Geist Al et al 1994) for more information about the functionality of and how to use the PVM.

6.5.2 Message passing interface

The Message Passing Interface (MPI) project started when a number of vendors of concurrent computers, researchers, government laboratories and other parties of industry, joined together to create the Message Passing Interface Forum (http://www.mpi-forum.org/). The object of the forum was to create a standard for an interface for message passing. The members of the group intended to create a practical, portable, efficient and flexible standard for message passing, and in 1992 the first draft was presented.

During the standardization of the MPI, the MPI Forum attempted to take advantage of previous experiences and adapt the most attractive features of previous message passing systems. The design of the MPI has thus been influenced by many previous projects.

The MPI Forum had a number of goals when constructing the MPI, some of which were to:

- Design an Application Programming Interface (API) that can and will be used by developers of parallel applications.
- Allow efficient communication by avoiding memory copying as well as allowing offloading communication to a communication co-processor if available.
- Allow implementations that can be used in heterogeneous environments.
- Assume reliable communication so that the developer does not have to worry about transmission errors.

The first, MPI Standard version 1.0, was released in 1994 and the standard became widely accepted. A number of free and commercial toolkits have been developed and this also leads to an increased user base. The MPI is recognized and supported by most vendors of concurrent computers.

A number of open-source MPI implementations have also been developed; MPICH (http://www-unix.mcs.anl.gov/mpi/mpich/) and LAM/MPI (http://www.lam-mpi.org/) are two of the most ambitious. They both have large user bases and are actively being developed. We shall describe MPICH, the toolkit used in the prototype of this project, in more detail below. More information on MPI programming can be found in (Marc Snir) and in the MPI Standard documents found on the website of the MPI Forum (http://www.mpi-forum.org/).

6.5.3 MPICH

The MPICH (http://www-unix.mcs.anl.gov/mpi/mpich/) is one of the most used implementations of the MPI. It is an open source project that aims at producing a highly portable implementation of the MPI Standard. Major parts of it are developed at the Argonne National Laboratory.
The MPICH attempts to follow the MPI Standard very closely and stay portable rather than optimize performance. Some of its features (as of November 2002) include:

- MPI Standard 1.2 compliance.
- Support for a wide variety of environments such as clusters of single processor computers, clusters of multi-processor computers or massively parallel computers.
- Parts of the MPI 2 Standard implemented.
- Parallel programming tools such as trace and log file creation as well as performance analyzer.

A communication layer called the Abstract Device Interface (ADI) was written as a communication framework. The ADI allows MPICH to be ported to different communication systems, and this enables MPICH to be optimized for different hardware such as high-speed network interfaces. Manufacturers of computer hardware can thus use the ADI to write communication drivers that optimize performance for specific hardware.

6.6 JOB SCHEDULING AND ALGORITHM DISTRIBUTION

Task distribution is an optimization problem much like maximum flow, minimal distance, etc. The object is to optimize some function such as the total execution time or the average completion time of a number of tasks. What function to use depends on the characteristics of the problem: the type of service required, how the problem is solved, how to measure solution quality, etc. The distribution algorithm used should optimize this function, but in reality it is impossible to use the optimal distribution algorithm (it is too expensive) and hence, approximate methods are used.
The idea is that, in some situations, it would be beneficial to use a more intelligent, (possibly expensive in terms of resources) distribution algorithm that can compensate its cost with improved solution quality. The gain of the algorithm should be higher than the cost of using it. If the tasks are homogeneous and the system has no predictable behavior that can be taken advantage of, a simple task distribution function is most likely good enough. If the tasks or the system have properties that can be taken advantage of, it may be possible to find a more intelligent distribution function.

6.6.1 Mainframe systems and job scheduling

Mainframe systems commonly run batch-jobs, and the job-scheduling algorithm based its calculations on each job’s parameters such as the memory and, CPU-time required, what devices and resources were required and other data such as priority. The scheduling algorithm decided the running order of the jobs, given this input.

Job scheduling is NP-complete both in common single-processor and multi-processor cases (Ausiello et al 1999), which means that approximate methods have to be used. Evaluating how well an algorithm performs can be done with the simulation data.

6.6.2 Distribution Algorithms

Distribution systems implement different distribution algorithms. Usually, the assumption is that we have a set of tasks (may be ordered in time) and a set of workers. Both the tasks and workers may have specific parameters that determine their behavior. The distribution algorithm should minimize some distribution function or be effective in some common situation. Some common algorithms are:
• Round-robin: The workers are logically ordered in a ring and the task assignment is decided by a orderly ring traversal. This algorithm will provide an even distribution.

• Weighted round-robin: This is the round-robin slightly modified so that workers may appear multiple times during one complete ring traversal. This allows for uneven distribution of the tasks to compensate for example workers with different levels of performance.

• Priority: An example of this algorithm would be to send tasks to one specified worker if available; otherwise one of the backup workers is selected.

• Least tasks: Keep track of how many tasks are being processed by each worker and send the current task to the worker processing the smallest number of tasks. Can also be used with TCP connections.

• Fastest response: Send the current task to the worker that provided the best response time on a recent task. The response time must be specifically defined, perhaps the time it took to respond to a recent request.

• Hash function: This algorithm uses some pieces of information (The hash key) from the task such as an identifier, and applies a hash function on this to determine the selected worker. The object of the function is to create an even or “almost random” distribution and the hash key as well as the hash function should be selected with this in mind.

Which algorithm to use depends on the tasks and the system. It is often very difficult to calculate the best algorithm to use, and empiric testing is often required.
6.7 DATA ACCESS

A program consists of logic and data. The logic is the code to be executed and the data is used during the execution. The data can be from input parameters, read from files, interactive information caused, network communication, etc. A distributed application might not have the data stored locally where it can be accessed fast with high bandwidth; thus, the data access service must be supplied efficiently to make sure the application can get its data quickly and safely.

Distributed file system

Distributed file systems are vital parts of many organizations and extremely important in computer systems. The design of a distributed file system is significant because of the large number of parameters and events, as well as the demands of a fault-tolerant system. Common problems that have to be solved are, how to handle data consistency, whether a state or stateless protocol should be used and how to make the system secure.

Some examples of distributed file systems are Network File System (NFS), Andrew File System (AFS) and AppleTalk. The File Transfer Protocol (FTP) and Hypertext Transfer Protocol (HTTP) are both variations of distributed file systems in the sense that they provide remote access to files, but they are less advanced when it comes to features such as file locking. Their simplicity has, however, made them popular, especially for simple file-transfers over the Internet.

Distributed applications and grid computing require a distributed file system with a set of properties such as being usable, allowing localization of files (by using mirrors) to enable faster access, allowing simple access from
multiple places and offering security for file access and transfer. The system must also provide functionality to handle different versions of a file to make sure that the data is not corrupted and each instance of an application reads the same data.

**Distributed database**

A database is often difficult to run in a distributed environment because some operations require locking of entries and tables; This is very expensive in a high-latency and low-bandwidth environment. There are a number of commercial distributed databases available and their design always tries to limit the effects of this problem. Usually a local cache is used on the nodes, but to make sure the data is consistent, the caches must be invalidated after a write; thus, many writes will cause bad performance.

Although a distributed database may, in general, be potentially bad in performance it can be quite efficient for certain applications, especially if the database reads are far more common than the database writes. The problem types are categorized based on the communication requirements of the problem in relation to its computation requirements.

### 6.8 EXPERIMENTATION

The recent growth in the volume of image data being generated and used for a variety of applications has led to the development of image databases. The richness of content and subjective interpretations of image has rendered text-based queries inadequate. CBIR is a new but widely adopted method for finding images from vast and un-annotated image databases. The correctness of retrieval for CBIR depends on efficient and effective indexing and searching schemes. Subjective queries and retrieval demands enormous
computation time due to large data sizes of images coupled with large and complex indices required for search. Networks of workstations (NOW) are a cost-effective way of providing the much needed computational power in such applications.

This section presents a distributed scheme for the classification and retrieval of images in an image database using the NOW system. It uses an initial classification and a heuristic for determining the average feature vectors and distance in the image classes. The results of classification, retrieval, speedup obtained and the correctness of the retrieval are presented. The results indicate the viability and effectiveness of the proposed scheme.

The drastic increase in the generation, processing and use of image data in a variety of applications has necessitated the development of image database systems to manage the massive amounts of image data. For such image databases, keyword-based indexing is inadequate and content-based indexing is essential. The reasons to opt for content-based indexing, are the inexact nature, subjective interpretations and difficulty in formulating exact keyword based queries. Also, it demands fast and accurate retrievals from the user’s perspective. Important factors required for achieving fast and accurate retrievals are (i) derivation of prominent features to be used as indices during search, (ii) organization of these indices in a suitable data structure and (iii) the measure of similarity corresponding to perceptual similarity. Indices which characterize the image data should be of low dimensionality. Such type of content-based retrieval produces a list of objects which are similar to the given query.

Several schemes have been proposed for image retrieval using content based queries (Bentley and Sedgewick et al 1998, Antony chan et al 1998, Norbert Sensen). The classification of images into groups of similar
images facilitates the improvement in the speed and accuracy of content-based retrievals. Adopting manual classification for such a voluminous database is a difficult task. Automated classification is a suitable alternative. However, it places enormous demands on computation and disk access.

NOW is one of the cost-effective ways of providing much needed computational power and exploiting the available under-utilized resources to meet such high computational needs. Software such as PVM (Parallel Virtual machine), and MPI (Message Passing Interface) provide the user with a unified view of a single machine. Generally, process and data migration is needed to distribute the load on the machines in a manner transparent to the user.

6.9 SYSTEM MODEL AND ASSUMPTIONS

The distributed systems consist of multiple CPUs with their own way of interconnection and communication. (Tanenbaum and Steen 2003). In this work, we used a collection of multi computers interconnected through a fast Ethernet and loaded with message passing APIs. The general structure of the model used in our work is given in Figure 6.2.

Each node has its own kernel, containing modules for managing local resources such as memory, the local CPU, and a local disk. Also, each has a separate module for handling inter processor communication, that is, sending and receiving messages to and from either of the node. Above each kernel is a common layer of software that implements the operating system as a virtual machine supporting parallel and concurrent execution of various tasks.
In this model, one machine is treated as a master and all other systems are used as slaves. The MPI is loaded in all the machines for communication between the machines. This architecture is effectively used for our experimentation. To classify large databases, a set of features is computed from the images. Wavelet-based statistical features are used as indices during classification and search operations. Images are classified using the Haar Wavelet (Liang and Chen 2004) and the resultant average component is further used to retrieve prominent Haralick features (Haralick 1979). Haralick had proposed 14 statistical features out of which 4 different features are considered in this work, namely, contrast, entropy, Angular Second Moment (ASM), and homogeneity. Subramanya et al (2002) used the distribution with color histogram features.

The next section describes the distributed algorithms for classification and retrieval with a few assumptions.
Assumptions:

- One of the machines in the NOW, designated as the master, has all the image data files.
- The data is analyzed and the feature vector is constructed for all the images before classification starts.
- The images are partitioned into clusters using (Hartigan 1975), (Jain and Dubes 1988) clustering algorithms.
- They are then distributed to the machines before the search starts.
- Access of the disk of machine $i$ by $j$ ($j \neq i$) if necessary, will not generally conflict with access to machine $i$.
- There is a shared array named balance of size $P$; balance [i] contains the number of images which remain to be compared to the query at machine $i$.

6.10 DISTRIBUTED CLASSIFICATION AND RETRIEVAL ALGORITHMS

It is assumed that all images have been analyzed and the feature vector has been built. Then the images are partitioned into $P$ (approximately) equal parts, and each partition is sent to a machine. The distance between the two images refers to the Euclidean distance between their corresponding feature vector parameters. The steps of the algorithm are described below: where each of the steps is done by every machine concurrently.
Step 1: Apply the Pair wise Nearest Neighbor (PNN) clustering technique to get an initial clustering of the images in the partition in each machine.

Step 2: Compute the average value for each of the initial classes. This value is equi-distant from all other images in the class.

Step 3: Send this average value of each machine to all the others.

Step 4: Recluster the images in each machine based on the average value in each machine. i.e, each image is compared with all the average values and clustered with the one with the least distance. Thus, at the end of the step, each machine will have P classes.

Step 5: Recompute the average value in each machine and send the new set of average value to all the machines.

Step 6: Determine the new average value of the classes which are the averages of the corresponding classes in all the machines.

Step 7: Check for convergence when the number of re-classification i.e. the amount of changes in the class size is less than a threshold; then the procedure is considered as having converged. If there is a convergence, go to the next step, otherwise go to step 4.

Step 8: Do a final merge and split of the classes across all machines. When the difference between the two average values of the classes is below a threshold, the two classes can be combined together into a single cluster. On the other hand, when a cluster is too sparse, then the cluster is further split into two classes further.
The PNN clustering technique (Arnold et al 2002) is used to derive the initial classes in each machine. The algorithm proceeds iteratively by finding a pair of images which are closest and merging them, and determining a new average feature which represents the class. This process is repeated until the number of classes reaches a desired size.

**Master**

Partition the image database data into “classes”
Distribute the classes among the slaves and broadcast the query to all slaves.

**Slave**

Determine the “average” and “distance” at all levels, for all the classes in the slaves. Determine the level (Resolution) of search
Do similarity matches for the given query in the class(es) in the machine.
Send a message to the console as and when a match is found, with the details of the match.

Receive the message about the match and append the match to the result list and sort it.

Determine if ‘considerable’ data is yet to be matched in (any machine); if so, determine the partition and send a message to the machine

Send a message to the console when all data has been used in matching.

Transfer of index data occurs if necessary, as determined by the console and the matching continues

Determine if the matching (searching) is over, and present the results.

**Figure 6.3 Sequence of actions at the console and the other machines**
In PNN clustering the Euclidean distance measure is used while merging classes. In our implementations, the distance between the two class feature vectors are computed using equation discussed in section 2.8.

6.10.1 Retrieval algorithm

Feature Vectors are organized as pattern matrices and are used as indices in the searches. The images are classified into classes of similar images using the clustering algorithm (Hartigan 1975, Jain and Dubes 1988) and the image classes are distributed to the machines. The query is then broadcast to all the machines. The average value and the distances are computed for the image classes in each machine. All the machines then carry out the search in the selected classes and send the result to the console. The sequence of actions at the master and the other slave in the NOW is shown in Figure 6.3. Algorithmic steps for query processing are given below.

**DISTRIMAGE RET** (in: I, Q; Out; R)
I: set of images, Q: Query, R: Retrieval results

1. Set up : Basic system Initialization
2. Distrimage Classify: This scheme automatically classifies the images using NOW.
3. Districlass: This distributes the classes to all machines.
4. Broadcastquery: Broadcasts the query to all the machines.
5. Do concurrently

   Each machine P does

   (i) Determine average feature (IP, Avg)

   (ii) Determine distance (IP, Avg, R)
(iii) Select classes (Avg, R, Q)

6. Do concurrently Console does:
   (i) Respond to messages from other machines
   (ii) If balance[i] = True,
        For all i then
           R ← best β results, exit
        End if
   Machines:
   (i) Image Search

7. Until all machines are done.

6.10.2 Performance Evaluation

The proposed algorithms have been tested to classify images in a database of about 5000 images. The database is built using (Brodatz 1966, Corel database). The NOW system used for the experiments consisted of five P-IV@ 1.7 GHz machines. The machines are connected by a 100Mb/sec Ethernet and running MPI. Four sample classes obtained using the proposed algorithms with minimal set are shown in Figure 6.4.

The classification time for the different number of images, using different number of machines is shown in Figures 6.5 (a), 6.5 (b) and the speedup of the proposed algorithm to a sequential scheme in 6.5 (c). It is easily seen that the speedup is linear in the number of active machines used in the classification process.

To evaluate the retrieval efficiency of the proposed system, we use the performance measure, Precision and Recall as defined in equation (5.8) and (5.9).
Figure 6.4 Sample image classes

Table 6.1 Computation time in sec, for Classification

<table>
<thead>
<tr>
<th>Number of images/Time in seconds</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>P=2</td>
<td>18</td>
<td>20</td>
<td>24</td>
<td>48</td>
</tr>
<tr>
<td>P=3</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>P=4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>SEQUENTIAL</td>
<td>22</td>
<td>31</td>
<td>42</td>
<td>75</td>
</tr>
</tbody>
</table>

Table 6.2 Speedups with and without classification

<table>
<thead>
<tr>
<th>Number of machines used in Search/Speedup</th>
<th>With classes</th>
<th>Without classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1.4</td>
<td>1.8</td>
</tr>
<tr>
<td>3</td>
<td>2.2</td>
<td>2.8</td>
</tr>
<tr>
<td>4</td>
<td>2.6</td>
<td>3.5</td>
</tr>
<tr>
<td>5</td>
<td>3.6</td>
<td>4.2</td>
</tr>
</tbody>
</table>
Figure 6.5  (a) Computation time in sec, for Classification (b) Speedups with Time taken Versus Number of machines (c) Speedups with and without classification

Table 6.3  Speedups with Time taken Versus Number of machines

<table>
<thead>
<tr>
<th>Number of machines used in Search/Speedup</th>
<th>Computation time (sec)</th>
<th>I/O time (sec)</th>
<th>Total time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1.18</td>
<td>1.25</td>
<td>1.24</td>
</tr>
<tr>
<td>3</td>
<td>1.19</td>
<td>1.18</td>
<td>1.26</td>
</tr>
<tr>
<td>4</td>
<td>1.22</td>
<td>1.24</td>
<td>1.33</td>
</tr>
<tr>
<td>5</td>
<td>1.14</td>
<td>1.15</td>
<td>1.35</td>
</tr>
</tbody>
</table>
The first six retrieval results for a given query are presented in Figure 6.6. The retrieval operation is performed both with and without classification. The Time taken for retrieval of the query image with classification is significantly less compared to the retrieval without classification.
The precision and recall calculations for a test set are plotted and the variations are shown in Figure 6.7 (a) and 6.7 (b). According to the classification techniques

<table>
<thead>
<tr>
<th>Number of clusters</th>
<th>Haralick Features without DWT</th>
<th>Haralick with DWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.350</td>
<td>0.568</td>
</tr>
<tr>
<td>6</td>
<td>0.460</td>
<td>0.515</td>
</tr>
<tr>
<td>7</td>
<td>0.370</td>
<td>0.487</td>
</tr>
<tr>
<td>8</td>
<td>0.350</td>
<td>0.431</td>
</tr>
<tr>
<td>9</td>
<td>0.316</td>
<td>0.431</td>
</tr>
<tr>
<td>10</td>
<td>0.305</td>
<td>0.427</td>
</tr>
<tr>
<td>11</td>
<td>0.282</td>
<td>0.395</td>
</tr>
<tr>
<td>12</td>
<td>0.321</td>
<td>0.376</td>
</tr>
<tr>
<td>13</td>
<td>0.316</td>
<td>0.386</td>
</tr>
<tr>
<td>14</td>
<td>0.302</td>
<td>0.353</td>
</tr>
<tr>
<td>15</td>
<td>0.333</td>
<td>0.349</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number of clusters</th>
<th>Haralick Features with DWT</th>
<th>Haralick without DWT</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.310</td>
<td>0.523</td>
</tr>
<tr>
<td>6</td>
<td>0.450</td>
<td>0.477</td>
</tr>
<tr>
<td>7</td>
<td>0.430</td>
<td>0.454</td>
</tr>
<tr>
<td>8</td>
<td>0.480</td>
<td>0.449</td>
</tr>
<tr>
<td>9</td>
<td>0.477</td>
<td>0.492</td>
</tr>
<tr>
<td>10</td>
<td>0.472</td>
<td>0.490</td>
</tr>
<tr>
<td>11</td>
<td>0.474</td>
<td>0.406</td>
</tr>
<tr>
<td>12</td>
<td>0.542</td>
<td>0.461</td>
</tr>
<tr>
<td>13</td>
<td>0.534</td>
<td>0.488</td>
</tr>
<tr>
<td>14</td>
<td>0.524</td>
<td>0.397</td>
</tr>
<tr>
<td>15</td>
<td>0.514</td>
<td>0.450</td>
</tr>
</tbody>
</table>
6.11 CONCLUSION

The high volume of images and the requirements for content-based retrievals in image databases place enormous demands on storage and computation. NOW is a cost-effective way of providing the much-needed computation power in such applications. This chapter presented a distributed scheme for the classification and retrieval of images based on the NOW system. This scheme has used the feature vector based on wavelet and Haralick features. The distance measures between the feature vectors of images in the database and the query image are used for ranking the retrieval results.

The speed up obtained is linear. This increases with the active machines used in the classification process. To validate the results of retrieval, performance measures such as precision and recall are calculated, and the values are promising.
This classification and the retrieval operations can be further improved by (i) Considering the images as band filter responses, (ii) Selection of most promising features according to the applications, (iii) Minimizing the disk access and bus conflicts, (iv) Application specific data sets (e.g. Medical Imaging, Geosciences processing, Military database, Remote sensing), and (v) Query Retrieval with relevance feedback.