Noname manuscript No.
(will be inserted by the editor)

Building a Web-scale Image Similarity Search System

Michal Batko · Fabrizio Falchi · Claudio Lucchese · David Novak · Raffaele Perego · Fausto Rabitti · Jan Sedmidubsky · Pavel Zezula

Received: date / Accepted: date

Abstract As the number of digital images is growing fast and Content-based Image Retrieval (CBIR) is gaining in popularity, CBIR systems should leap towards Web-scale datasets. In this paper, we report on our experience in building an experimental similarity search system on a test collection of more than 50 million images. The first big challenge we have been facing was obtaining a collection of images of this scale with the corresponding descriptive features. We have tackled the non-trivial process of image crawling and extraction of several MPEG-7 descriptors. The result of this effort is a test collection, the first of such scale, opened to the research community for experiments and comparisons. The second challenge was to develop indexing and searching mechanisms able to scale to the target size and to answer similarity queries in real-time. We have achieved this goal by creating sophisticated centralized and distributed structures based purely on the metric space model of data. We have joined them together which has resulted in an extremely flexible and scalable solution. In this paper, we study in detail the performance of this technology and its evolvement as the data volume grows by three orders of magnitude. The results of the experiments are very encouraging and promising for future applications.

Keywords similarity search · content-based image retrieval · metric space · MPEG-7 descriptors · peer-to-peer search network

1 Introduction

It is common in database community to open a discussion with some reference to the data explosion. According to recent studies, we will create more data in the next three years than has been produced in all of human history. But where is all this data

M. Batko · D. Novak · J. Sedmidubsky · P. Zezula
Masaryk University, Brno, Czech Republic
E-mail: [batko, david.novak, xsedmid, zezula]@fi.muni.cz

F. Falchi · C. Lucchese · R. Perego · F. Rabitti
ISTI-CNR, Pisa, Italy
E-mail: [fabrizio.falchi, claudio.lucchese, raffaele.perego, fausto.rabitti]@isti.cnr.it

coming from? The Enterprise Strategy Group[1] estimates that more than 80 billion photographs are taken each year. To store them would require 400 petabytes of storage. If an average digital photo occupies 750 KB, it takes as much space as 30 pages of digital text, i.e., about 18,000 words. The management of digital images promises to emerge as a major issue in many areas providing a lot of opportunities in the next couple of years, particularly since a large portion of pictures remains as "unstructured data".

Current searching engines headed by Google are in the center of current information age; Google answers daily more than 200 million queries against over 30 billion pages. However, the search power of these engines is typically limited to text and its similarity. Since less than 1 % of the Web data is in textual form, the rest being of multimedia/streaming nature, we need to extend our next-generation search to accommodate these heterogeneous media. Some of the current engines search these data types according to textual information or other attributes associated with the files. An orthogonal and well-established approach is the Content-based Image Retrieval (CBIR). However, the problem of Web-scale CBIR does not seem to be solved in a satisfactory manner. The current publicly-available systems for large-scale CBIR index typically hundreds of thousands of images. There is a high discrepancy between these numbers and the volumes of images available on current Web, so we decided to investigate the situation by shifting the current bounds up by two orders of magnitude.

This work results from a cooperation of two research groups within the European project SAPIR[2] and it builds on experience and results accumulated over a long-term research in the area of CBIR and especially similarity indexing and searching. In the context of the SAPIR project, which aims at finding new content-based methods to analyze, index, and search large amounts of speech, image, video, and music, we have intended to develop a large-scale architecture for indexing and searching in image collections according to visual characteristics of their content. The system should be able to scale to the order of tens, or even hundreds, of millions. Until now, no image collection with respective descriptive features of such size has been available. The first contribution of this work is a report on the nontrivial goal of crawling a set of over 50 million images from a photo-sharing system Flickr[3] and extraction of five MPEG-7 features from every image. This collection, called CoPhIR Test Collection [1], has now been made available to the research community.

We use a specific combination of the visual features [3] to get an effective similarity measure for generic photo images. This combination can be effectively modeled by a metric space [27] and we have developed efficient indexing and searching mechanisms based purely on this data model. Another contribution of this work is a description of the route from indexing tens of thousands of images up to tens of millions and especially a unique comparison of performance experiments conducted in individual steps. The results show that we have achieved our goal and the final system synergically combining centralized and distributed search techniques can manage at least tens of millions of images while answering similarity queries in real-time. All our indexing and searching technologies were created within the MUFIN project [2]. Since the metric approach is highly versatile, all these mechanisms can be straightforwardly applied to various data types, for example, face recognition, music, video clips, etc.

[1] http://www.enterprisestrategygroup.com/

[2] EU IST FP6 project 045128: Search on Audio-visual content using Peer-to-peer IR

[3] http://www.flickr.com/

The rest of the paper is organized as follows. Section 2 maps the current state in the area of large-scale CBIR and it marks out the challenges we decided to face in this paper. Section 3 describes the process of building the test collection: crawling the images and extraction of the MPEG-7 features. Section 4 maps the evolution and performance of our indexing and searching mechanisms. Finally, Section 5 analyzes the results obtained, discussing also future research directions opened by our achievements.

2 Objectives

Before we specify the objectives and quantitative challenges of our work we mention the current systems for large-scale CBIR which are taken as a stepping stone.

2.1 Current State

As documented by a very recent and comprehensive survey [10], the endeavour after analyzing and searching digital images lasts for fifteen years, at least. Focusing on the large-scale systems for image retrieval, there is a number of publicly-available systems which base the search on annotations or other attributes associated with the images. In this case, traditional well-established indexing and searching techniques can be applied in order to efficiently manage Web-scale image collections: Google Images, Yahoo!, ExaLead[4], or PicSearch[5]. Because these systems may suffer from a lack of trustworthy image annotations, there are attempts to increase the quality of image labeling via interactive public-domain games, e.g. The ESP Game[6] or Google Image Labeler[7]. Searching photos by the geographic locations where they were taken is another example of attribute-based search. This functionality is provided, e.g. by Flickr system[8], which allows users to query about 30 million images in this way.

An orthogonal approach is adopted by CBIR techniques which search the images according to their visual content similarity. Such techniques typically extract a characterization (signature) of the image content, which is then used for indexing and searching. A majority of theoretical works and their applications [10, 25] are specialized and tuned for a specific application domain and/or they manage relatively small data collections. Speaking about large-scale systems, the Tiltomo project[9] indexes about 140,000 images from Flickr and searches them by color and texture characteristics; it also allows to combine it with "subject" searching. The ImBrowse[10] system allows to search a collection of about 750,000 images by color, texture, shapes and combinations of these (employing five different search engines). The Cortina system [12] searches over 10 million images by means of three MPEG-7 descriptors which are stored in flat file structures. The authors provide no technical details nor mention any response times or other performance characteristics. Recently, the Idée Inc. company has started a

[4] http://www.exalead.com/
[5] http://www.picsearch.com/
[6] http://www.espgame.org/
[7] http://images.google.com/imagelabeler/
[8] http://www.flickr.com/map/
[9] http://www.tiltomo.com/
[10] http://media-vibrance.itn.liu.se/vinnova/cse.php

product[11] which searches a commercial database of 2.8 million images. No details of the data, search technology or system performance could be found.

Another research stream is an automatic image annotating based on the analysis of the image content. System ALIPR [16] uses a training set of images which is separated into a number of concept categories; these categories are annotated. The indexed images are categorized and annotated by means of extracting visual features from them and by applying statistical methods. Searching is then based purely on the assigned annotations. The ALIPR has a public demo[12]; its internal functionality is provided by a system SIMPLIcity [26].

A great research effort is still necessary in the area of CBIR: different approaches have to be tried and evaluated. It is important to evaluate their user-perceived effectiveness as well as their performance on large-scale volumes. In this direction, it is worth mentioning the activities of several European research projects, in the field of multimedia search, integrated in the Chorus[13] initiative. Still, a high discrepancy remains between the data volumes which can be searched by content efficiently and the billions of images available on the Web nowadays.

2.2 Scalability Challenge

Regarding the quantitative characteristics of current search engines as mentioned above, we have established our scalability challenge as follows:

- to incrementally build, index, and test image collection up to 100 million images;
- the search time for main memory indexes ≤ 1 s;
- the search time for disk-based indexes up to 2–3 s.

These objectives go far beyond the current practice. On the data acquisition level, this would require to download and process from 15 TB to 50 TB of data, depending on the image resolution. Moreover, we need a storage space for the image descriptors (including the MPEG-7 features) and the thumbnails of about 1.5 TB. We also have to bear in mind that the image crawling and feature extraction process would take about 12 years on a single standard PC and about 2 years using a high-end multi-core PC.

Since the content-based retrieval is much more expensive than the text search, such data volume needs a non-trivial computational infrastructure support and it is necessary to apply scalable mechanisms to use the infrastructure effectively. By analogy to data acquisition, having one single-processor PC, evaluation of a similarity query on an image content index with a sequential scan would take over 10 hours, if the index is on disk, and about one hour, if the index could fit in main memory.

3 Building the Image Collection

Collecting a large amount of images for investigating CBIR issues is not an easy task, at least from a technological point of view. The challenge is mainly related to the size of the collection we are interested in. Shifting state-of-the-art bounds of two orders of magnitude means building a 100 million collection, and this size makes it very complex

[11] http://labs.ideeinc.com/
[12] http://www.alipr.com/
[13] http://www.ist-chorus.org/

to manage every practical aspect of the gathering process. However, since the absence of a publicly available collection of this kind has probably limited the academic research in this interesting field, we tried to do our best to overcome these problems. The main issues we had to face were:

1. identification of a valid source;
2. efficient downloading and storing of such a large collection of images;
3. efficient extraction of metadata (MPEG-7 visual descriptors and others) from the downloaded images;
4. providing reliable data-access to metadata.

In the following we will discuss the above issues by describing the challenges, the problems we encountered, and the decisions we took.

3.1 Choosing the Data Source

Crawling the Web is the first solution if you are looking for a practically unlimited source of data. There are plenty of images on the Web, varying in quality from almost professional to amateur photos, from simple drawings to digital cartoons.

There are also many different ways to retrieve such data. The first option is to exploit spider agents that crawl the Web and download every image found on the way. Of course, this would result in a large amount of time and bandwidth wasted in downloading and parsing HTML pages, possibly gathering only a few images. The authors of [6] report that the average number of images hyperlinked by HTML pages is varying. In their experiments with the Chilean Web, they repeatedly downloaded each time about 1.3 million Web pages. The number of images retrieved were 100,000 in May 2003, 83,000 in August 2003 and 200,000 in January 2004. Thus, assuming that these percentages are still valid today, we can expect that:

> ***Fact:*** To gather 100 million images, we would have to download and parse from 650 million to 1.5 billion Web pages.

A second option, which may be more efficient, is to take advantage of the image search service available on most commercial Web search engines. Just feeding the search engine with queries generated synthetically, or taken from some real query log, would provide us with plenty of images.

This abundance and diversity of Web images is definitely a plus. Not only because we want a large collection, but also because we want our collection to spread over different kinds of images. A problem is instead given by the large differences in the quality and size of the retrieved images. A large portion of them are decoration elements like buttons, bullet list icons, and many other which are very small images or photo thumbnails. These images are not suitable for our purposes and would pollute the corpus, some of them could be filtered out automatically as the feature extraction software is likely to fail on images with non-standard sizes.

However, for the need of high-quality data, we finally decided to follow a third way: crawling one of the popular photo sharing sites born in the last years with the goal of providing permanent and centralized access to user-provided photos. This approach has several advantages over the aforementioned approaches.

Image Quality In fact photo sharing sites like Flickr, PhotoBucket, Picasa, Kodak EasyShare Gallery, Snapfish, etc. mainly store high-quality photographic images. Most of them are very large since they come from 3–8 Megapixel cameras, and have a standard 4:3 format.

Collection Stability These sites provide quite static, long term and reliable image repositories. Although images may be deleted or made private by the owners, this happens quite rarely. Most photos stay available for a long time and they are always easily accessible. Conversely, the Web is much more dynamic, images change or are moved somewhere else, pages are deleted and so on.

Legal Issues The above consideration is very important also when considering the legal issues involved in the creation of such collection of images. In fact, storing for a long time a publicly available image may in some case violate author's copyrights. We are mainly interested in the visual descriptors extracted from the images, but any application of CBIR has to access the original files for eventually presenting the results retrieved to a human user. Since Photo sharing sites are fairly static, we can build a quite stable collection without permanently storing the original files, but maintaining only the hyperlinks to the original photos that can be accessed directly at any time.

Rich Metadata Finally, photo sharing sites provide a significant amount of additional metadata about the photos hosted. The digital photo file contains information about the camera used to take the picture, the time when it was taken, the aperture, the shutter, etc. More importantly, each photo comes with the name of the author, its title, and a description, often with user-provided tags. Sometimes also richer information is available such as comments of other users on the photo, the GPS coordinates of the location where the photo was taken, the number of times it was viewed, etc.

Among the most popular photo sharing sites, we chose to crawl Flickr, since it is one with the richest additional metadata and provides an efficient API[14] to access its content at various levels.

3.2 Crawling the Flickr Contents

It is well known that the graph of Flickr users, similarly to all other social media applications, is scale free [15]. We thus exploited the small-world property of this kind of graphs to build our huge photo collection. By starting from a single Flickr user and following friendship relations, we first downloaded a partial snapshot of the Flickr graph. This snapshot of about one million distinct users was crawled in February 2007. We then exploited the Flickr API to get the whole list of public photo IDs owned by each of these users. Since Flickr Photo IDs are unique and can be used to unequivocally devise a URL accessing the associated photo, in this way we have easily created a 4.5 GB file with 300 million distinct photo IDs.

In the next step, we decided what information to download for each photo. Since the purpose of the collection is to enable a general experimentation on various CBIR research solutions, we decided to retrieve almost all information available. Thus, for each

[14] http://www.flickr.com/services/api/

photo: title and description, identification and location of the author, user-provided tags, comments of other users, GPS coordinates, notes related to portions of the photo, number of times it was viewed, number of users who added the photo to their favourites, upload date, and, finally, all the information stored in the EXIF header of the image file. Naturally, not all these metadata are available for all photos. In order to support content based search, we extracted several MPEG-7 *visual descriptors* from each image [18]. A visual descriptor characterizes a particular visual aspect of an image. They can be, therefore, used to identify images which have a similar appearance. Visual descriptors are represented as vectors, and the MPEG-7 group proposed a distance measure for each descriptor to evaluate the similarity of two objects [17]. Finally, we have chosen the five MPEG-7 visual descriptors described below [17,13]:

Scalable Color It is derived from a color histogram defined in the Hue-Saturation-Value color space with fixed color space quantization. The histogram values are extracted, normalized and non-linearly mapped into a four-bit integer representation. Then the Haar trasform is applied. We use the 64-coefficients version of this descriptor.

Color Structure It is also based on color histograms but aims at identifying localized color distributions using a small structuring window. We use the 64-coefficients version of this descriptor.

Color Layout It is obtained by applying the DCT transformation on a 2-D array of local representative colors in Y, Cb, or Cr color space. This descriptor captures both color and spatial information. We use the 12-coefficients version of this descriptor.

Edge Histogram It represents local-edge distribution in the image. The image is subdivided into 4×4 sub-images, edges in each sub-image are categorized into five types: vertical, horizontal, 45° diagonal, 135° diagonal and non-directional edges. These are then transformed in a vector of 80 coefficients.

Homogeneous Texture It characterizes the region texture using the mean energy and the energy deviation from a set of 30 frequency channels. We use the complete form of this descriptors which consists of 62 coefficients.

There are several other visual descriptors in the MPEG-7 standard which can be useful, for example, for specialized collections of images (e.g. medical). Our experience [3] suggests that these five descriptors perform quite well on non-specialized images, such as the ones in our collection.

Unfortunately, the extraction of MPEG-7 visual descriptors from high-quality images is very computationally expensive. Although the MPEG-7 standard exists for many years, there is not an optimized extraction software publicly available. To extract descriptors, we used the MPEG eXperimentation Model (MPEG-XM) [13] that is the official software certified by the MPEG group that guarantees the correctness of the extracted features. This software running on a AMD Athlon XP 2000+ box takes about 4 seconds to extract the above five features from an image of size 500×333 pixels. Therefore, even without considering the time needed to download the image and all additional network latencies involved, we can estimate that:

Fact: A single standard PC would need about 12 years to process a collection of 100 million images.

It was thus clear that we needed a large number of machines working in parallel to achieve our target collection of 100 million images in a reasonable amount of time. For

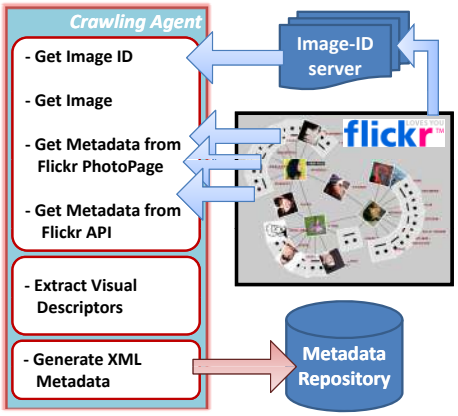

Fig. 1 Organization of the crawling and feature extraction process.

this reason, we developed an application that allows to process images in parallel on an arbitrary (and dynamic) set of machines. This application is composed of three main components: the *image-id server*, the *crawling agents*, and the *repository manager* as shown in Figure 1.

The image-id server was implemented in PHP as a simple Web application accomplishing the task of providing crawling agents with an arbitrary number of photo identifiers not yet processed.

The crawling agent is the core part of our application. It loops asking the image-id server for a new set of image identifiers to process. Once it obtains a set from the server, it starts the actual retrieval and feature extraction process. Given a photo ID, the first step is to issue an HTTP request and download the corresponding *Flickr photo-page*. This is parsed to retrieve the URL of the image file and some of the metadata discussed above. Thanks to Flickr APIs, this metadata is then enriched with other information (title of the photo, description, tags, comments, notes, upload date, user name, user location, GPS coordinates, etc.).

We downloaded medium-resolution version of the photos, which have the larger dimension of 500 pixels. This improves the independence of extracted features from image size and reduces the cost of processing large images. The MPEG-XM [18] software is used to extract the aforementioned five visual descriptors.

The extracted features and all the available metadata are used to produce an XML file containing the knowledge we collected from the image. Finally, a thumbnail is also generated from the photo. The XML file and the thumbnail of the image are sent to a Web-service provided by the *repository manager*.

The repository manager runs on a large file-server machine providing 10 TB of reliable RAID storage. In addition to receive and store the results processed by the crawling agents, the repository manager also provides statistic information about the state of the crawling process and basic access methods to the collection.

Fact: Disks provide a potentially unreliable storage and, actually, two disks had to be replaced.

Fig. 2 Machines collaborating on the crawling.

3.3 Using the GRID for Crawling and Feature Extraction

We have considered GRID to be the right technology to obtain large amount of computing power we needed. GRID is a very dynamic environment that allows to transparently run a given application on a large set of machines. In particular, we had the possibility to access the EGEE European GRID infrastructure provided to us by the DILIGENT IST project[15].

We were allowed to use 35 machines spread across Europe (see Figure 2). We did not have an exclusive access to these machines and they were not available all the time. Both hardware and software configurations were heterogeneous: they had various CPUs, memory, disk space, but also the libraries, software (e.g. Java), and Linux versions installed. Thus, we had to build a self-contained crawling agent.

The crawling agent is logically divided into two modules. The first one accomplishes the communication with the image-id server, crawls Flickr website, uses Flickr APIs, and sends the result of the computation to the repository manager. This was coded in Java to improve portability. However, since we could not assume the presence of the Java virtual machine on every machine, we incorporated into the crawling agents also a JVM and the required Java libraries. Due to the latencies of the crawling task, the crawling agent can instantiate a number of threads, each of them taking care of processing a different image. The setting, which proved to be adequate, has four threads per agent (per one CPU core) and processes a maximum of 1,000 images. These parameters induced computations times of 20 to 60 minutes depending on the CPU speed.

[15] http://www.diligentproject.org/

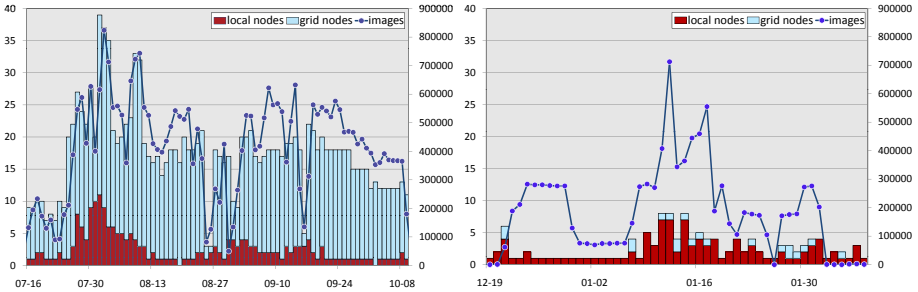

Fig. 3 Number of GRID and local machines available during the two crawling periods: July 16^{th} to October 9^{th} 2007 (left) and December 19^{th} 2007 to February 7^{th} 2008 (right).

The second module of the crawling agent is the MPEG-XM feature extraction software. Since the MPEG-XM software is not maintained, it has become incompatible with the recent compilers and libraries versions. For this reason and for the heterogenity of the GRID, we encapsulated into the crawling-agents also all the libraries it uses.

Submitting a job to a GRID infrastructure, the user does not have a full control on the time and location where the job runs. The GRID middleware software accepts the job description and schedules it on the next available machine according to internal policies related to the load of each node, the priority of the different organizations using the GRID infrastructure, etc. In our case, the job always first downloads the crawling-agent package from our repository-manager and then runs the software contained in the package. The GRID provides a best-effort service, meaning that a job submitted to the GRID may be rejected and never executed. Indeed, there are several factors that may cause the failure of a job submission.

Fact: From the 66,440 jobs submitted, only 44,333 were successfully executed that means that 33,3 % of the jobs failed due to unavailability of GRID resources.

Our straightforward approach together with the self-scheduling of images by each crawling agent has two important advantages. First, in case the GRID middleware is not able to deploy the given job, there would be no consequences in the remainder of the system, especially, no image will be skipped. Second, in case of a software update, it is just needed to replace the old version on the repository manager with the new one.

Not all of the GRID machines were available through the crawling period and, therefore, we also used a set of local machines in Pisa which processed the images during the GRID idle time. We thus reached the total of 73 machines participating in the crawling and feature extraction process.

The crawling process took place in two separate periods, both because of GRID availability and because we needed to consolidate the data after the first period. In Figure 3, we report on the number of machines available during the crawling process. During the first period, the GRID provided an average of 14.7 machines out of the 35 and, simultaneously, there were 2.5 local machines available, on average. Also the availability of the machines during the day was unstable: The local machines were mainly available over night while some of the GRID machines were available only for a few hours per day. During the second period, only one powerful multiprocessor machine

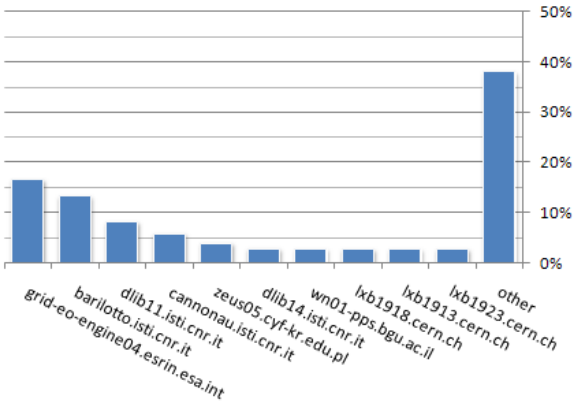

Fig. 4 Number of images processed by each site.

was available from the GRID, and we could continue the process only with our local resources.

Figure 4 reports the total number of images processed by each site. The best machine (provided by the GRID) processed about 17 % of the whole collection – this is a very powerful machine equipped with seven quad-core CPUs. The second best is a local machine used only during the second phase, which is equipped with two quad-cores Intel Xeon 2.0 GHz and it processed about 13 % of the collection. These machines were the most powerful and the most constantly available over time. However, the largest total contribution came from a number of machines each of which was able to process only a small number of images.

3.4 The CoPhIR Test Collection

The result of this complex crawling and image processing activity is a test collection that served as the basis of the experiments with content-based image retrieval techniques and their scalability characteristics, in the context of the SAPIR project.

We have not yet reached the target of 100 million images: we have made a checkpoint at about half of the target. The current image test collection has the following quantitative characteristics:

- number of images: 54.58 million (about 200,000 were removed by the data cleaning process);
- storage space: 245.3 GB for image descriptors, 54.14 GB for the image content index, 355.5 GB for thumbnails;
- on average, each photo is described by 3.1 textual tags, has been viewed 42 times by Flickr users, and received 0.53 user comments.

Given the effort required in building such test collection, and the potential interest to the international research community, to make experiments in large-scale CBIR, we decided to make it available outside the SAPIR project scope.

The result is the CoPhIR (Content-based Photo Image Retrieval) Test Collection, managed by ISTI-CNR research institute in Pisa. The data collected so far represents

the world largest multimedia metadata collection available for research purposes, with the target to reach 100 million images till the end of 2008. Each entry of the CoPhIR collection is an XML structure containing:

- link to the corresponding image on the Flickr Web site;
- the thumbnail of the photo image;
- the photo textual metadata: author, title, location, GPS, tags, comments, views, etc.;
- an XML sub-structure with 5 standard MPEG-7 visual descriptors.

Note that our use of the Flickr image content is compliant to the most restrictive Creative Commons license. Moreover, the CoPhIR test collection complies to the European Recommendation 29-2001 CE, based on WIPO (World Intellectual Property Organization) Copyright Treaty and Performances and Phonograms Treaty, and to the current Italian law 68-2003. The scientific organizations (universities, research labs, etc.) interested in experiments on CoPhIR have to register at the CoPhIR Web site[16] and to sign the CoPhIR Access Agreement establishing conditions and terms of use for the collection.

4 Content-based Image Retrieval

Building a large-scale system for efficient image similarity search is a tough and exciting challenge to face. Such system would put to test proclamations about the theoretical scalability of solutions on which the system is built. This section maps the route we followed from thousands of images towards tens of millions.

We perceive the content-based retrieval problem as a triangle: the type of *data* and the way the similarity is assessed, *indexing techniques* which are used to enhance the efficiency, and *infrastructure* the system is running on. The data we index and search is formed by the five MPEG-7 features described above; we model this data as a single metric space. All index and search techniques are based purely on the metric space model and are implemented over the same framework. The index structures and their implementations are very flexible regarding the hardware infrastructure they run on.

4.1 Metric Space Approach

The metric space model [27] treats the data as unstructured objects together with a function to assess proximity of pairs of objects. Formally, the *metric space* $\mathcal{M}$ is a pair $\mathcal{M} = (\mathcal{D}, d)$, where $\mathcal{D}$ is the *domain* of objects and d is the total *distance function* $d : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$ satisfying the following postulates for all objects $x, y, z \in \mathcal{D}$:

$$
\begin{array}{ll}
d(x,y) \geq 0 & (\textit{non-negativity}); \\
d(x,y) = 0 \quad \text{iff} \quad x = y & (\textit{identity}); \\
d(x,y) = d(y,x) & (\textit{symmetry}); \\
d(x,z) \leq d(x,y) + d(y,z) & (\textit{triangle inequality}).
\end{array}
$$

The semantics of this concept is: The smaller the distance between two objects is, the more similar they are. The metric space is typically searched by similarity queries

[16] http://cophir.isti.cnr.it/

Table 1 The features with their respective distance measures and their weights in the aggregate metric function.

MPEG-7 Feature	Metric	Weight
Scalable Color	L_1 metric	2
Color Structure	L_1 metric	3
Color Layout	sum of L_2	2
Edge Histogram	special	4
Homogeneous Texture	special	0.5

defined by a query *object* $q \in \mathcal{D}$ and some *constraint* on the data to be retrieved from the indexed dataset $I \subseteq \mathcal{D}$. The basic type of similarity query is the *range query* $R(q, r)$ which retrieves all objects $o \in I$ within the range r from q (i.e., $\{o \,|\, o \in I, d(q, o) \leq r\}$). From the user point of view, the most important query is the *nearest neighbors query* $kNN(q, k)$ which returns the k objects from I with the smallest distances to q. This type of query is also convenient for an image search system and we will focus on it in the following.

Fact: Metric space is a highly flexible model of similarity.

The metric space model provides us with a unique generality and almost absolute freedom in defining and combining the similarity measures. Specifically, as the individual MPEG-7 features and their distance functions form metric spaces, we can combine several features into a single metric function by a weighted sum of the individual feature distances. The particular distances are normalized before being weighted and summed. Table 1 summarizes the features we use, their respective distance measures [17], and their weights in the aggregate metric function.

4.2 Implementation

All indexing and searching structures described below have been implemented over a unified Metric Similarity Search Implementation Framework (MESSIF) [8] and are written in Java. The MESSIF contains the encapsulation of the metric space model and provides a number of modules from a basic storage management, over a wide support for distributed processing, to automatic collecting of performance statistics. Due to its open and modular design, the integration of individual components is straightforward. The MESSIF offers several generic clients to control the index structures and there is also a customized GUI for image similarity search.

4.3 100K: Centralized Searching

Our first indexing experience with the Flickr images was with a rather small dataset of 100,000 images. We have used the M-Tree [9] technique which is de-facto a standard for similarity searching based on metric technology. Similarly to B-Trees and R-Trees, all objects are stored in (or referenced from) leaf nodes while internal nodes keep pointers to nodes at the next level, together with additional information about their subtrees.

Specifically, an object called pivot and a covering radius is used to specify the minimal sphere-like region of the metric space the respective subtree covers. The M-tree is a dynamic structure, thus the tree can be built gradually as new data objects come in. The insertion algorithm looks for the best leaf node in which to insert a new object and, if there is not enough space, the node is split possibly propagating the split to higher tree levels.

Since the original M-Tree is practically a decade old, there are many extensions available focusing on various improvements. We have decided to implement the Pivoting M-Tree (PM-Tree) extension [22], which employs additional filtering by means of precomputed distances between all objects stored in the structure and a fixed set of pivots, which improves search space pruning. We also apply an advanced node-splitting algorithm defined in Slim-Tree [24] in order to keep the M-Tree regions as compact as possible.

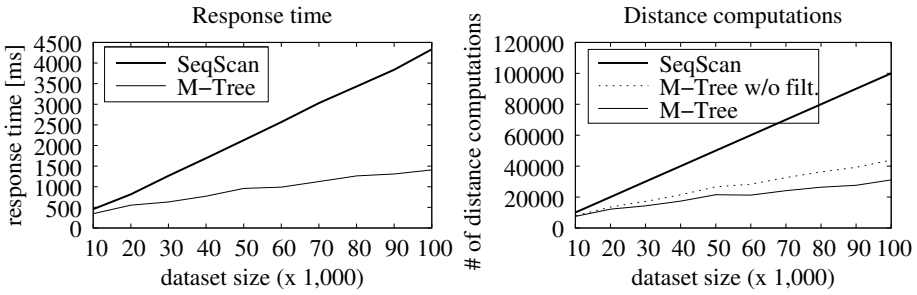

Fig. 5 Response times and number of distance computations of M-Tree for $kNN(q, 50)$.

Experience

Figure 5 reports on our experience with the M-Tree. To analyze the scalability trends, we have inserted the 100,000 objects in ten batch series of 10,000 objects. After each batch, we have run a set of fifty $kNN(q, 50)$ queries with different query objects q and studied the evolution of response times (left graph). To establish a baseline, we have compared our results with a sequential scan algorithm. It is a simple implementation of the *kNN* search that computes the distance to the query object q for all the objects one by one and keeps the k best. Our implementation also confirmed the well-known fact that a query evaluation spends most of the time in metric function computations. Thus, we have measured this cost during our trials and we report on our findings in the right graph. To show the improvement gained by the PM-Tree filtering, the graph shows also the number of distance computations without this extension (denoted as "M-Tree w/o filt.").

The graphs prove that M-Tree enhances the search performance significantly. However, we can also see that the response times grow linearly as we increase the size of the dataset. It means that we can use the M-Tree structure for indexing several tens of thousands images, but we cannot go much farther if we require on-line responses.

Fact: Centralized indices achieve online response for up to 100,000 images.

4.4 1M: Distributed Searching

The previous section indicates that the similarity search is inherently expensive and has identified the practical boundaries of the centralized structures for similarity search. A natural way to overcome the limitations of the centralized resources is to shift towards distributed environments. Such approach allows to exploit parallelism during the query processing and also provides an easily enlargeable and virually unlimited storage capacity.

Our similarity distributed structures [7] are based on the paradigm of Structured Peer-to-Peer (P2P) Networks. The system consists of *peers* – nodes which are equal in functionality. Each peer manages a part of the overall dataset and maintains a piece of navigation information which allows to route a query from any peer to the one with relevant data. In the case of similarity indices, the query is typically routed to multiple peers which search their local data and compute partial answers. The originating peer then gathers all the partial answers and merges them into the total result of the query.

The M-Chord Structure

The M-Chord [21] is a P2P data structure for metric-based similarity search which is used in our system. The M-Chord has been inspired by a centralized vector-based method called *iDistance* [14] – they both map the data space into a one-dimensional domain in a similar way. The M-Chord then applies any one-dimensional P2P protocol, like Chord [23] or Skip Graphs [5], in order to divide the data domain among the peers and to provide navigation within the system. The search algorithms exploit the features of the data-mapping to navigate the queries only to relevant peers.

The M-Chord mapping (schematically depicted in Figure 6a) works basically as follows: Several reference points are selected globally from the sample dataset (we call these objects *pivots* and are denoted as p_i). The data space is partitioned in a Voronoi-like manner into *clusters* C_i (each object is assigned to its closest pivot). Following the iDistance idea, the one-dimensional mapping of the data objects is defined according to their distances from the cluster's pivot. The M-Chord key for an object $x \in C_i$ is defined as

$$mchord(x) = d(p_i, x) + i \cdot c$$

where c is a *separation constant* greater than any distance in the dataset. To evaluate a range query $R(q, r)$, the data space to be searched is specified by *mchord* domain intervals in clusters which intersect the query sphere – see an example in Figure 6b.

Having the data space mapped into the one-dimensional M-Chord domain, every active node of the system takes over responsibility for an interval of keys. The range queries are routed to peers responsible for the intervals as seen in Figure 6b. The algorithm for nearest neighbors query $kNN(q, k)$ consists of two phases: (1) The query is evaluated locally on "the most promising peer(s)"; in this way, we obtain an upper bound r_k on the distance to the k^{th} nearest object from q. (2) Range query $R(q, r_k)$ is executed and the k nearest objects from the result are returned. There is a natural tradeoff between the cost of the first phase and the precision of the r_k estimation and, thus, cost of the second phase. We use an experimentally tuned setting which visits several peers in the first phase. For more details about M-Chord and its performance see [7, 21].

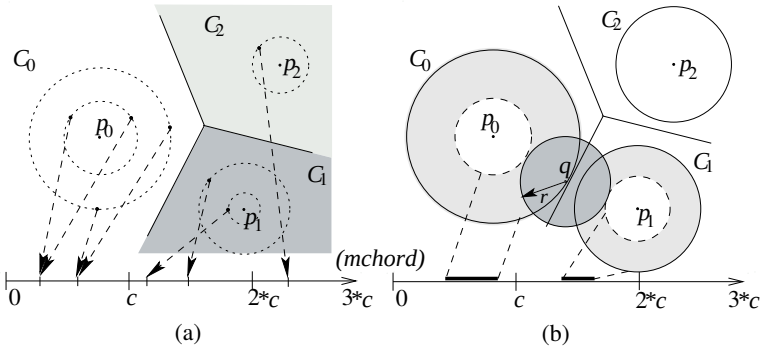

Fig. 6 M-Chord mapping and search principles.

In our system, the local data of each peer is organized in an M-Tree structure (see Section 4.3). The overall architecture of the system is schematically depicted in Figure 7. In experiments throughout this section, the M-Chord uses 20 pivots to partition the data space into clusters and the peer-to-peer navigation protocol is Skip Graphs.

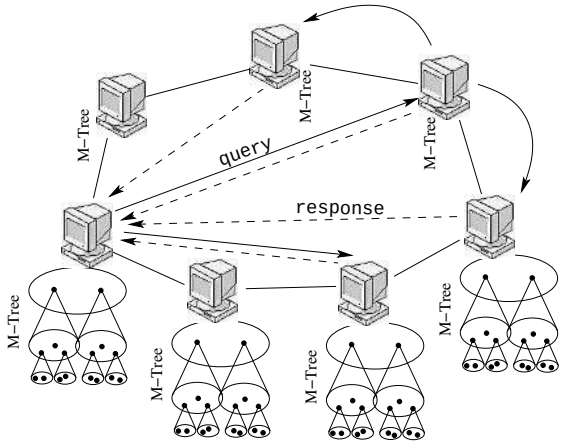

Fig. 7 Schema of the distributed system.

Experience

The experimentation with a distributed system requires an appropriate hardware. Our system is based purely on peer-to-peer paradigm and our implementation is strictly detached from the actual physical hardware being written in Java and using TCP/IP communication protocol. Thus, we are able to run the system on wide variety of infrastructures. Our 1M images network consists of 50 peers mapped to 16 CPUs. In the experiment, we have run a batch of *kNN* queries varying the k, i.e., the number of most similar images returned to the user. The results for each k are averages over 50 different query images.

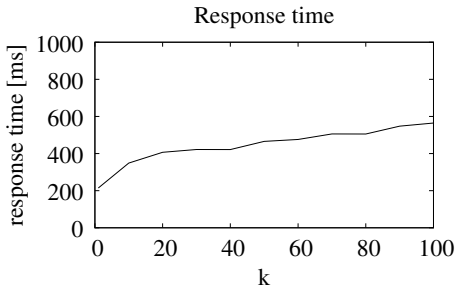

Fig. 8 Response times of 1M network.

Figure 8 shows the response times of the described experiment. We can see that the search times increase only slightly with the number of nearest objects (k). The response times about 500 ms are achieved by two facts: (1) each peer organizes approximately 20,000 objects in a local M-Tree index and (2) evaluation of the queries on the relevant peers proceeds in parallel, whenever allowed by the hardware infrastructure.

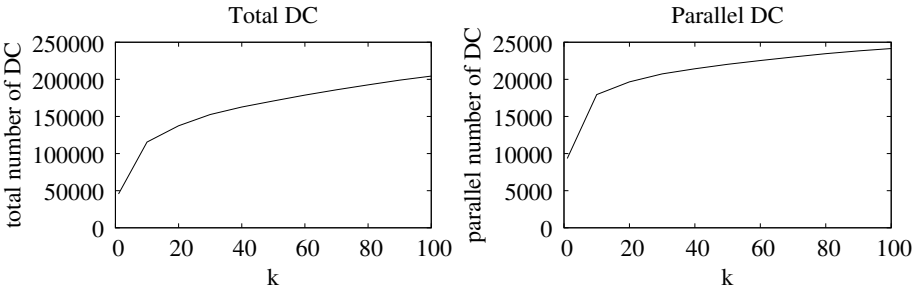

Fig. 9 Number of distance computations of 1M network.

Fact: We achieve 500 ms response times on the one million image collection with 50 peers on 16 CPUs.

Figure 9 shows the number of distance computations corresponding to this experiment. The "total number of distance computations" (left graph) expresses the theoretical costs of the searching if the whole distributed system shared a single CPU. In the distributed environment, the query processing is done in parallel and the "parallel costs" (right graph) represent the maximal number of distance computations evaluated on a single peer. This graph shows a general (theoretical) value of parallel DC independent of the actual number of CPUs and of other hardware infrastructure details. The difference among the total and parallel costs is proportional to the number of peers accessed to get the answer, which is about 30 peers in this case, which is shown in Figure 10 (left graph).

The right graph in Figure 10 reports on the number of messages sent during the query processing. Note that messages are needed for both contacting the destination peers and for responses from the peers which consult their local index. In our case, ap-

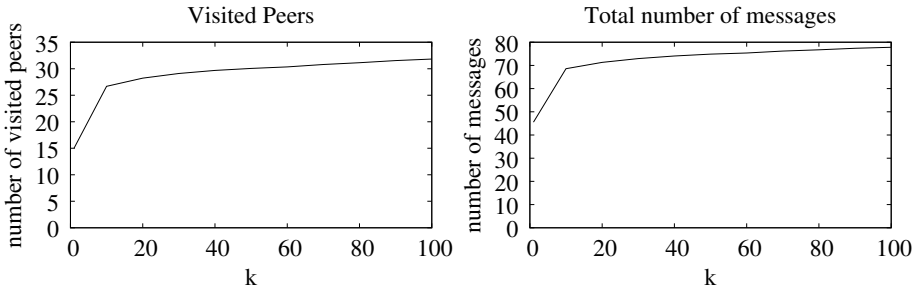

Fig. 10 Number of visited peers and messages exchanged.

proximately 44 messages were needed to contact the 31 peers and another 31 messages were sent back to the originating peer for the *kNN* queries with $k = 100$. The contribution of the communication to the response time is typically negligible compared to the evaluations of the distance function.

4.5 10M: Approximate Distributed Searching

The previous section showed that we can retain online response times for a 1M dataset having an appropriate hardware infrastructure. We tried to increase the utilization of the hardware and build a 10M network on the same hardware configuration. The previous section also shows that *kNN* queries accessed over half of the network's peers. Having a higher number of data objects, which makes the metric space denser, the answers are more compact and thus the percentage of visited peers is lower. Despite this fact, the absolute volume of the data accessed by the query is disproportionately higher than the number of objects returned by the query. We have uncovered that majority of the answering peers contribute to the answer result only negligibly or not at all and they are accessed only to prove the preciseness of the answer. We decided to allow a moderate approximation and thus prune the number of accessed peers significantly. We have reached this goal and thus managed to run the 500-peers network with 10M images on a relatively modest hardware configuration.

Approximation in M-Chord

The approximate evaluation-strategy for the *kNN* queries in M-Chord [19] is based on the *relaxed branching* policy [4]. The basic idea is to examine highly-promising data partitions only and ignore those with low probability of containing qualifying data. Thus, the most promising parts of the most promising clusters are explored in M-Chord. The clusters are selected using a tunable heuristic that takes into consideration the distance between the query and the cluster's pivot. Within a selected cluster, the query is first navigated to the peer where the query object would be stored, if it belonged to the cluster, and then its neighboring peers are accessed. We can tune the query costs versus the quality of the approximation by specifying the number (or percentage) of peers visited within each cluster. A thorough experimental analysis of the M-Chord approximation [19] confirmed a very high performance gain with a moderate or even no degradation of the answer quality.

Approximation in M-Tree

The approximation at the M-Chord level allows us to limit the number of peers visited during a search. We could also decrease the response times of the local search by exploiting approximate algorithms also in local M-Trees. The strategy [28] is based on a stop condition embedded into its *kNN* evaluation strategy. The M-Tree's *kNN* traversal uses a priority queue where the tree nodes are sorted according to their potential to contain objects satisfying the specified query. The stop condition allows us to quit the search even if the queue is not empty after a certain portion of data has been explored.

M-Tree Split

The creation of the peer-to-peer network, i.e., to actually insert the data into the index, turned out to be a challenge in itself. Building the 1M network from the scratch required about 40 minutes which is feasible even if the network is recreated repeatedly. However, more than ten hours required for the 10M was unacceptable, so we have investigated the building bottlenecks. The network grows as new peers join the network, which always requires that some peer splits its local data. By now, the M-Tree was split by deleting objects from the original M-Tree and building a new M-Tree by inserting the objects one-by-one. We have proposed an algorithm [11] which allows to split an instance of M-Tree according to a given metric-based condition (either defined by a generalized-hyperplane or ball partitioning). Basically, the M-Tree is traversed from the root to its leaves and the partitioning condition is applied to the covering ball regions (in inner nodes) and to objects (in leaves). As a result, some branches of the tree can be separated without any computations and some can be split very efficiently using pre-computed distances already stored in the M-Tree. The experiments confirmed [11] that the algorithm outperforms the baseline delete and insert strategy about 28 times. With this optimization, we were able to shorten the building time of 10M network to about two hours.

Experience

We have shifted the number of stored objects by one order of magnitude, which in terms of hardware means ten times higher demands on space. Fortunately, our 16-CPU infrastructure offers 64 GB RAM in total so we can still keep all the 500 peers in main memory. The M-Chord approximation algorithm picks 1–7 clusters to be accessed by the query (2.5 clusters on average); we set the percentage of peers visited in each of these clusters to 40 %. The local approximate search in the M-Trees was set to explore maximally 30 % of data.

We have repeated the experiment with the same queries as for the 1M dataset using both the precise and approximate *kNN* search. The number of visited peers and response times are reported in Figure 11. Comparing the precise query results with the 1M network (in Figure 10 left) roughly six times more peers are visited while the dataset is ten times bigger – this confirms our expectation that the metric space is more dense. Moreover, the response times increased only four times (see Figure 8) even though we are using the same number of CPUs. This is mainly because of some fixed overhead incurred in each query regardless of the network size. Also the mapping

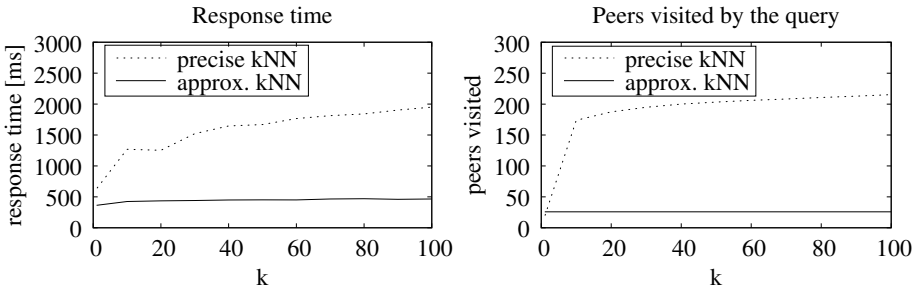

Fig. 11 Response times and number of visited peers of the 10M network.

of 50 peers to 16 CPUs does not utilize all the CPUs fully in every query evaluation while the load is more uniformly distributed for 500 peers.

Fact: Approximate similarity queries on the 10M distributed index are answered in 500 ms using 16 CPUs.

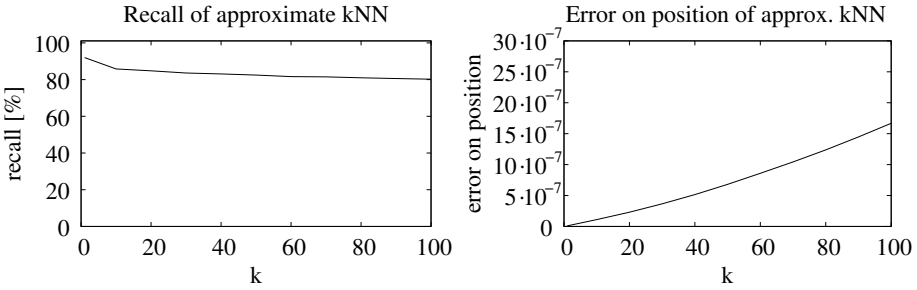

Fig. 12 Recall and relative error on position of the approximated search.

The graphs in Figure 11 also show that the approximated search reduced the number of visited peers to 25 peers and thus kept the response times about 500 milliseconds. The quality of the approximate answers is captured in Figure 12. The left graph shows the recall of the approximation, i.e., the percentage of the precise answer that is returned by the approximate query. We can see that our approximation setting gives more than 80% of the precise answer.

The right graph in Figure 12 shows another approximation quality measure, the relative error on position [27], which exposes the positions of objects missed by the approximation. More precisely, the measure is computed as a sum of differences of object positions in precise and approximate answers (both ordered by distance from the query object). The value is normalized by dividing the number by the dataset size and k. Taking an example of $k = 20$, a value $40/(20 \cdot 10^7) = 2 \cdot 10^{-7}$ means that each of the returned object in the approximate answer on a 10^7 dataset is, on average, shifted by two positions. We can also see that the most similar objects, i.e., k is small, have practically zero error on position thus the approximation always finds the best-matching objects and make mistakes only on higher positions.

Fact: Approximate searching cuts the costs by one order of magnitude while preserving a high answer quality.

4.6 50M: Disk-oriented Distributed Searching

Currently, the final scale we have practical experience with is a distributed structure indexing 50 million images. The network consists of 2,000 peers each with approximately 25,000 objects. In order to test the scalability of our system, we temporarily gained access to a hardware infrastructure five times larger than for the 10M network (i.e., 80 CPUs and about 250 GB RAM). These resources were available for short periods of time and thus we needed a bulk-loading mechanism for preparation of individual peers on other resources.

Simultaneously, we experimented with a persistent storage to cut the memory demands. In this implementation, the peers keep the leaf nodes of the M-Trees on disk, reducing the overall memory usage of the 2,000-peers network to 80 GB RAM; recall that the implementation is written in Java, which is relatively memory demanding and the programmer does not have a full control over the memory administration. Experiments with the disk-oriented were conducted on infrastructure with 32 physical disks and 32 CPUs. The objects stored on the disks occupied about 100 GB.

Bulk Loading

Because our 50M dataset is known in advance, the partitioning schema can be precomputed. Each object from the dataset can be assigned an M-Chord key (see Section 4.4) which determines the peer where the object will be stored – recall that each peer is responsible for an interval of M-Chord keys. The bulk-loading procedure thus proceeds as follows. First, an M-Chord key is computed for each object from the dataset and the dataset is sorted according to these keys (a disk-oriented merge sort). Then, the ranges of M-Chord keys are selected for the peers by splitting the sorted dataset into 25,000 objects parts. At this stage, each peer knows its M-Chord key range and we can build the M-Chord navigation structures and establish links between the peers. Finally, we take the peers one by one and fill their internal M-Trees with data from the respective dataset part. We are using Java serialization that allows us to store a running process to disk and restore it later on another computer.

Fact: Bulk loading allows to build networks of practically unlimited size on limited resources.

Persistence Bucket

In the previous steps, we have used a memory-only implementation, because we had enough RAM. The MESSIF [8] library offers a file-storage area for objects and our M-Tree implementation can use this functionality to become fully disk-oriented. On the other hand, our experiments revealed that the internal nodes of our 25,000-object M-Trees occupy approximately 8% of the size of the whole M-Tree. Thus, we decided to keep the inner nodes in memory effectively reducing the disk accesses to only sequential

scans of the leaf nodes that are hit during the search. Our experiments show that by this decision, we have gained a noticeable *kNN* search times boost at the cost of 5–10 MB of RAM per M-Tree.

Experience

As the 50M data space is denser than the 10M set, we can afford to use more restrictive settings of the approximation – the query visits only 20 % of the most promising peers of each cluster (the number of clusters is again determined by the same heuristic). The left graph in Figure 13 depicts the recall of this approximation settings as the k grows. We can see, that the recalls are only slightly lower than for 10M, where we accessed 40 % of each cluster (see left graph in Figure 12). With this settings, we expected the number of visited peers to be approximately doubled as for the 10M network (see the approximate line in Figure 11, right graph) as we have four times more peers now and visit one half of the clusters' percentage. This expectation has proven true – the queries visited about 53 peers, on average.

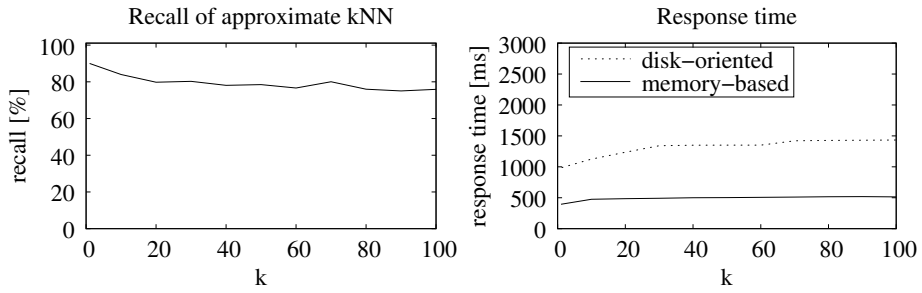

Fig. 13 Recall and response time for 50M network.

The results of the memory-based and the disk-oriented implementation differ only in the response times achieved – the comparison is in the right graph in Figure 13. With the memory-based implementation, we have reached practically the same response times as with the 10M network (see Figure 11). We can see that the disk-oriented implementation has approximately triple response times. Let us realize, that these times can be further improved as they are strongly influenced by the hardware settings (the number and speed of the disks, usage of disk arrays, etc.) and also by the quality of the implementation.

Fact: The experiments with the 50M collection confirmed that, having enough resources, the system is fully scalable.

4.7 The Scalability Story

To summarize our indexing scalability story described in Section 4, we show all the gathered response times of the various approaches for different dataset sizes in Table 2. For convenience, we provide also some estimated values (marked with star). These

are rough lower bound estimates, because these values ignore the additional overhead incurred by the bigger data as well as memory constraints – in these estimations, we assume that the whole dataset fits in memory.

Table 2 Response times of $kNN(q, 50)$ using various techniques on different dataset sizes.

Technique	**CPU**	**100k**	**1M**	**10M**	**50M**
Sequential scan	1	4.3s	43.4s	7.2m*	36m*
M-Tree	1	1.4s	12s	1.8m*	-
Parallel sequential scan	16	-	2.7s	27s*	2.3m*
	80	-	-	5.4s*	27s*
M-Chord	16	-	0.45s	1.7s	-
M-Chord with approximation	16	-	-	0.44s	-
	80	-	-	-	0.45s
M-Chord with approximation and disk	32	-	-	0.87s	1.4s

We can see that using a naive approach of sequential scan, the 10M dataset would require more than 7 minutes to process one query. Even a parallel sequential scan, where the whole dataset is divided into 16 disjoint parts that are searched by one CPU each, would require 27 seconds to complete assuming that all the data are in main memory and neglecting the overhead of merging the partial answers. The situation is even worse for our current 50M dataset, where even parallel work of 80 CPUs would not evaluate a single query below half a minute. Our experiments have proved that using a correct technology we can build a working system even on huge database of 50 million images that maintains online response times of about one second.

Contrary to the sequential scan, which fully utilizes all the CPUs for a single query, the distributed system does not require all of them all the time. Although we have not thoroughly studied the throughput of the system, i.e., the ability to serve multiple users at the same time, our experience says that the 10M network on 16 CPUs can answer up to 10 queries in parallel and the bigger 50M data using 80 CPUs can process more than 30 concurrent queries per second on average.

5 Conclusions

No doubts that the scalability problem for new digital data types is real, which can be nicely illustrated by difficulties with the management of the fast growing digital image collections. In this paper, we focus on two strictly related challenges of scalability: (1) to obtain a non-trivial collection of images with the corresponding descriptive features, and (2) to develop indexing and searching mechanisms able to scale to the target size.

We have crawled a collection of over 50 million high-quality digital images, which is almost two orders of magnitude larger in size than most existing image databases used for content-base retrieval and analysis. The images were taken from the Flickr photo-sharing system which has the advantage of being a reliable long-term repository of images and which offers quite a rich set of additional metadata. Using a GRID technology, we have extracted five descriptive features for each image. The features are defined in MPEG-7 standard and express a visual essence of each image in terms of colors, shape, and texture. This information is kept handy in XML files – one for each

Fig. 14 Example of three results of $kNN(q, 7)$ query in collections with different data volumes.

image – together with the metadata and links to original images in Flickr. We also store thumbnails of the original images, so the search engines built for this dataset can quickly show the overview of the results. This unique collection [1] is now opened to the research community for experiments and comparisons.

We have also proved that our distributed content-based retrieval system can scale to tens of millions of objects. The system offers a search for visually-similar images giving answers typically below one second on the database of 50 million images. An online demonstration of the system, currently indexing 100 million images, is available from the MUFIN web page [2]. A demonstration of MUFIN was recently accepted for the ACM SIGIR conference [20]. The presented technology is highly flexible as it is based on the metric space model of similarity and on the principles of structured peer-to-peer networks. In this paper, we report on several technological advances we have employed and various problems we had to solve on our route to indexing 50 million images. We provide results of numerous experiments measured on a real hardware infrastructure in real conditions. In addition, we have also created a Web interface which allows regular users to interact with the system and actually search for similar images.

An example of three query results is shown in Figure 14. Images similar to the query image (shown on top) were looked up in 1 million, 10 million and 50 million datasets. The figure illustrates a noticeable improvement in effectiveness, i.e., the search quality, whenever the dataset grew by one order of magnitude. We have observed this improvement practically for any query which can be explained by the fact that the search space is becoming denser, thus presence of similar (more close) images is more likely. For this reason, we expect this trend to continue for even bigger files. Another way of improving the effectiveness is to use more sophisticated image descriptors. Since we have mainly focused on the scalability problems in this paper, our decision was to take the descriptors of the well-established MPEG-7 standard.

In the future, we plan to investigate the application of additional descriptive features of the images, such as local features, combining visual features and user defined tags, and research the relevance feedback to further improve effectiveness. In addition,

we would like to test the technology also on other multimedia types such as video, speech, and music.

Acknowledgements This research was supported by the EU IST FP6 project 045128 (SAPIR) and national projects GACR 201/08/P507, GACR 201/09/0683, GACR 102/09/H042, and MSMT 1M0545. Hardware infrastructure was provided by MetaCenter[17] and by IBM SUR Award.

References

1. CoPhIR (Content-based Photo Image Retrieval) Test Collection (2008). URL http://cophir.isti.cnr.it/
2. MUFIN (Multi-Feature Indexing Network) (2008). URL http://mufin.fi.muni.cz/
3. Amato, G., Falchi, F., Gennaro, C., Rabitti, F., Savino, P., Stanchev, P.: Improving image similarity search effectiveness in a multimedia content management system. In: Proc. of Workshop on Multimedia Information System (MIS), pp. 139–146 (2004)
4. Amato, G., Rabitti, F., Savino, P., Zezula, P.: Region proximity in metric spaces and its use for approximate similarity search. ACM Transactions on Information Systems (TOIS) **21**(2), 192–227 (2003)
5. Aspnes, J., Shah, G.: Skip graphs. In: Proc. of ACM-SIAM Symposium on Discrete Algorithms, pp. 384–393 (2003)
6. Baeza-Yates, R.A., del Solar, J.R., Verschae, R., Castillo, C., Hurtado, C.A.: Content-based image retrieval and characterization on specific web collections. pp. 189–198 (2004)
7. Batko, M., Novak, D., Falchi, F., Zezula, P.: On scalability of the similarity search in the world of peers. In: Proc. of INFOSCALE, Hong Kong, pp. 1–12. ACM Press, New York, NY, USA (2006)
8. Batko, M., Novak, D., Zezula, P.: MESSIF: Metric similarity search implementation framework. In: Proc. of DELOS Conference, *LNCS*, vol. 4877, pp. 1–10 (2007)
9. Ciaccia, P., Patella, M., Zezula, P.: M-Tree: An efficient access method for similarity search in metric spaces. In: Proceedings of VLDB'97, August 25–29, 1997, Athens, Greece, pp. 426–435 (1997)
10. Datta, R., Joshi, D., Li, J., Wang, J.Z.: Image retrieval: Ideas, influences, and trends of the new age. ACM Computing Surveys **40**(2), 1–60 (2008). DOI http://doi.acm.org/10.1145/1348246.1348248
11. Dohnal, V., Sedmidubsky, J., Zezula, P., Novak, D.: Similarity searching: Towards bulk-loading peer-to-peer networks. In: 1st International Workshop on Similarity Search and Applications (SISAP), pp. 1–8 (2008)
12. Gelasca, E.D., Guzman, J.D., Gauglitz, S., Ghosh, P., Xu, J., Moxley, E., Rahimi, A.M., Bi, Z., Manjunath, B.S.: Cortina: Searching a 10 million + images database. Tech. rep., University of California, Santa Barbara (2007)
13. ISO/IEC: Information technology - Multimedia content description interfaces. Part 6: Reference Software (2003). 15938-6:2003
14. Jagadish, H.V., Ooi, B.C., Tan, K.L., Yu, C., Zhang, R.: iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. ACM Transactions on Database Systems (TODS 2005) **30**(2), 364–397 (2005). DOI http://doi.acm.org/10.1145/1071610.1071612
15. Kumar, R., Novak, J., Tomkins, A.: Structure and evolution of online social networks. In: KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowedge Discover and Data Mining, pp. 611–617. ACM Press (2006)
16. Li, J., Wang, J.Z.: Real-time computerized annotation of pictures. In: MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia, pp. 911–920. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1180639.1180841
17. Manjunath, B., Salembier, P., Sikora, T. (eds.): Introduction to MPEG-7: Multimedia Content Description Interface. John Wiley & Sons, Inc., New York, NY, USA (2002)
18. MPEG-7: Multimedia content description interfaces. Part 3: Visual. ISO/IEC 15938-3:2002 (2002)

[17] http://meta.cesnet.cz

19. Novak, D., Batko, M., Zezula, P.: Web-scale system for image similarity search: When the dreams are coming true. In: Proceedings of the Sixth International Workshop on Content-Based Multimedia Indexing (CBMI 2008), p. 8 (2008)
20. Novak, D., Batko, M., Zezula, P.: Generic similarity search engine demonstrated by an image retrieval application. In: Proc. of the 32st ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), Boston, USA. ACM (2009). Accepted for publication.
21. Novak, D., Zezula, P.: M-Chord: A scalable distributed similarity search structure. In: Proc. of INFOSCALE, Hong Kong, pp. 1–10. ACM Press (2006)
22. Skopal, T., Pokorný, J., Snásel, V.: PM-tree: Pivoting metric tree for similarity search in multimedia databases. In: Proc. of ADBIS, Budapest, Hungary (2004)
23. Stoica, I., Morris, R., Karger, D.R., Kaashoek, F.M., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. of SIGCOMM, San Diego, CA, pp. 149–160. ACM Press (2001). DOI http://doi.acm.org/10.1145/383059.383071. URL citeseer.ist.psu.edu/article/stoica01chord.html
24. Traina Jr., C., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-Trees: High performance metric trees minimizing overlap between nodes. In: Proc. of EDBT, *LNCS*, vol. 1777, pp. 51–65. Springer-Verlag (2000)
25. Veltkamp, R.C., Tanase, M.: Content-based image retrieval systems: A survey. Tech. Rep. UU-CS-2000-34, Department of CS, Utrecht University (2002)
26. Wang, J.Z., Li, J., Wiederhold, G.: SIMPLIcity: Semantics-sensitive integrated matching for picture libraries. IEEE Transactions on Pattern Analysis and Machine Intelligence **23**(9), 947–963 (2001). DOI http://dx.doi.org/10.1109/34.955109
27. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach, *Advances in Database Systems*, vol. 32. Springer-Verlag (2006)
28. Zezula, P., Savino, P., Amato, G., Rabitti, F.: Approximate similarity retrieval with m-trees. VLDB Journal **7**(4), 275–293 (1998)