A Distributed Image-Database Architecture for Efficient Insertion and Retrieval*

Roger Weber **H.-J. Schek**
Institute of Information Systems
ETH Zentrum, 8092 Zurich, Switzerland
{weber,schek}@inf.ethz.ch

Abstract

Allowing for content-based image retrieval, an image database must perform a number of time-consuming feature extraction tasks when inserting a new image, when searching with a newly entered image, or when bulk-loading many images in parallel. The duration of this preprocessing step typically lies far beyond what one would accept for conventional (short-running) database transactions. Hence, a natural step is to export the extraction tasks from the database and to distribute them among a number of external components in a cluster. As a consequence the database plays the role of a coordinator that assigns tasks to "its" auxiliary components. Since the execution times for feature extraction and the availabilities of such components vary over time, the coordinator needs additional information about the components in order to "optimally" assign extraction tasks to available components such that the overall preprocessing cost is minimized. At the same time, similarity search should be supported with high priority. In this paper, we present a coordination middleware (CoMid) on top of a distributed, component-based architecture that minimizes the cost of the preprocessing step and optimally assigns extraction tasks. We provide a solution for a dynamic environment: Any time a new feature extraction component can be added and any time a component can be switched off. Insertions as well as searches will not need to be restarted. The coordination middleware takes care of such "failures" and initiates "forward recovery". We describe in detail what data on the components and what image meta-data are necessary for the coordination, and how to exploit them. We report on an evaluation of this architecture using our prototype system applied to a very large image collection.

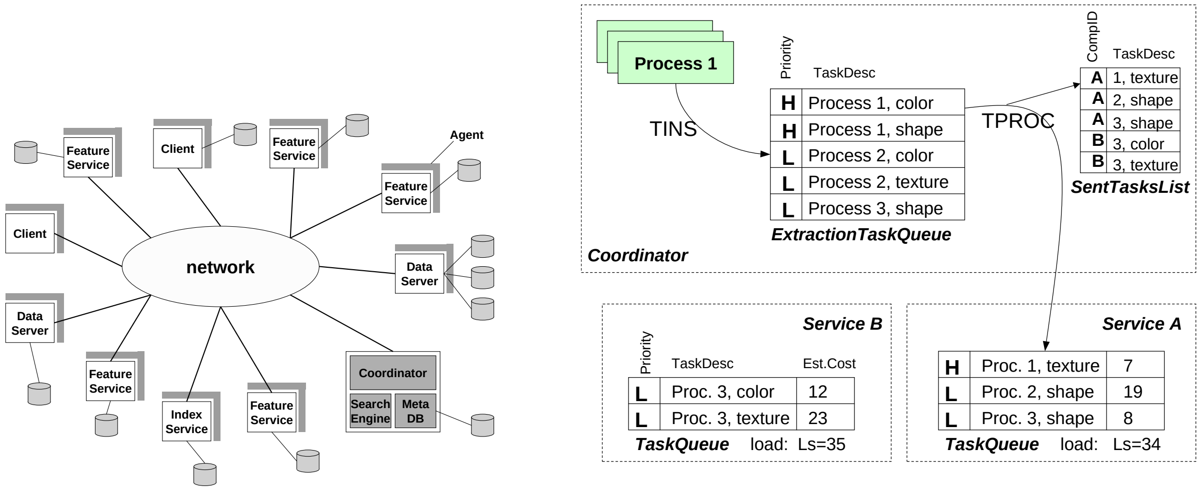
1 Introduction

Content-based image retrieval is based on the idea of abstracting the contents of an image by so-called features like color or shape. Instead of comparing the images on the raw data level, one only considers the feature representations of the images. Two images are then similar if their feature representations are *close* to each other. If the features are represented by points in a high-dimensional vector space, closeness of two feature representations means nearness of the points in the vector space. From another perspective, employing features for content-based retrieval, one *approximates* the contents of the raw image data with a much smaller feature representation (e.g. a vector), and uses only these approximations to identify similar objects. Hence, the amount of searched data is drastically reduced when performing a similarity search. Over the last years, large effort has been put in the development of better feature representations [6, 8, 9, 13, 12, 5] and of better indexing techniques [1, 7, 2, 4, 15].

A serious problem of an image search and retrieval system is the cost of extracting features. This step is necessary when new images are inserted or when a similarity search with a new image must be supported. For example, the texture descriptors specified by Manjunath and Ma [9] requires the application of several Fast Fourier Transformations (FFT) to the image data. Even for moderate-sized images, it takes several seconds to compute this feature representation. The extraction cost even grows further, if more than one feature type is employed by the image search system. It is obvious that due to this expensive preprocessing, a centralized image database system will soon become a bottleneck if it performs the feature extraction itself. Rather, it looks much more promising to utilize as many external components to preprocess the images and to determine the various feature representations.

In this paper, we follow this principle and we even go a step further and also let the image data remain in external repositories. Consequently, the central image database maintains only *meta-data* on

*This work has been partially funded in the framework of the European ESPRIT project HERMES (project no. 9141) by the Swiss *Bundesamt für Bildung und Wissenschaft* (BBW, grant no. 93.0135), and partially by the ETH cooperative project (no. 41-2641.5) on *Integrated Image Analysis and Retrieval.*



(a) (b)

Figure 1: (a) Architectural overview. (b) Maintenance of the extraction task queue.

the images stored externally, and on the status of external components and services such as storage components (data server), feature extraction components (feature service or FEE), indexing components (index service), and a coordinator component running the coordination middleware (see Figure 1 (a)).

Given this sort of obvious architectural environment consisting of the coordination middleware (CoMid) and of many specialized components, several questions arise in the context of preprocessing images: How many components for feature extraction should be applied? How can we cope with unreliable components? How can we dynamically add or remove components without restarting the system or without disconnecting clients or restarting bulk-load processes? How can we support higher priority tasks such as the similarity search by a client while the system inserts new images in parallel or while the system is in a bulk-load step? How do we cope with failures, i.e. how fast can we detect that a component is no longer available and how do we react? How can we optimally assign the extraction tasks for new images to the available components in order to minimize the overall preprocessing cost? What data about the components and about their dynamically varying status must be collected and how do we utilize them?

Out of these large collection of problems, the overall architecture and its justification are contained in [3]. The VA-file for the indexing components and search parallelization is described in [15, 14]. The aspects of failures treatment (atomicity) and concurrent process management are presented in [11, 10]. In the following, we concentrate on the coordination of the feature extraction components and on the "optimal" assignment of extraction tasks to the available components. We have implemented a coordination middleware using existing up-to-date database technology such as short transactions, 4GL stored procedures and triggers accessing the dynamically changing meta-information describing all components and their status. We see the contribution of this paper in the description, evaluation and experimentation of this highly distributed architecture and as a justification of it.

The structure of the paper is as follows: In Section 2 we give an overview of the general architecture of the CoMid, and describe in detail how extraction tasks are distributed among the available components. Section 3 discusses performance aspects, and Section 4 concludes.

2 CoMid-Architecture for Preprocessing Step

2.1 Architectural Overview

All components shown in Figure 1 (a) except the coordinator have a so-called agent process attached to it. An agent observes the component and informs the coordinator about the component. For this purpose the agents and the coordinator have a local database for the storage of meta-data, information on components and status information. All local databases are concurrently accessed by the related agent and by the coordinator. The main advantages of such a distributed architecture are good scalability, autonomy for data and service providers, and extensibility in terms of data and services (e.g. new feature types, new indexing schemes). Coordinating the components is a complex task with regard to the processes we want to support, namely inserting an object, similarity search and bulk-loading of objects. Consider for example the insertion of a new image (process InsertObject). Once a data provider adds a new image to its repository, the system must determine the feature values for the new image (with the help of feature extraction services), must update some of the external feature indexes (by requesting remote index services), and has to insert the meta-information into the meta-database. The coordination middleware (CoMid) provides correctness and guaranteed termination for distributed processes by employing transactional process management [11, 10]. While the database system provides us with the basic transactional guarantees on the level of a single step in a process, the middleware layer guarantees semantic isolation and semantic atomicity on the higher-level of concurrent processes [10].

2.2 Requirements for Preprocessing

When we insert an image (InsertObject-process), it is necessary to invoke not only one feature extraction task but many different ones resulting in a number of feature values for the specified image(s). The same happens in case of a SimilaritySearch (with the reference image not in the database), UpdateObject (with changed image data), and in case of the upload of an entire repository (Bulkload-process). In the following, we use the term *preprocessing step* to denote the entirety of extraction tasks for a given process. The task of the CoMid is to distribute the extraction tasks of all current processes among the available services such that:

1. SimilaritySearch processes are preferred,
2. the time required for a single InsertObject- or UpdateObject-process is minimal,
3. the tasks of a Bulkload-process optimally exploit the available services, and
4. the influence of a service failure is minimal, i.e. no process is blocked.

Subsequently, we describe the distribution scheme deployed by our CoMid.

2.3 Necessary Meta-Information for Preprocessing Step

Information gathered by the agents: When a feature extraction service is installed, its agent registers the service at the meta-database. The registration contains information about which features the service can extract, about how many processors and how much memory the service has, and about how costly the extraction of the available features is. The cost parameters are derived as follows: The agent measures the time for the extraction of feature values for some images with varying sizes. Based on these measurements, the agent develops a model for the extraction cost as a function of the image size. In our experimental environment, we used a simple linear model which turned out to be accurate enough[1]: The only model parameter is the time per pixel (`tpp`) spent for the extraction. Thus, the cost for an extraction task is determined by: $cost = num.of pixels \times \texttt{tpp}$. Furthermore, the parameter `tpp` indicates the fastness of a service. As such, it serves the coordinator to decide which service is fastest for a specific extraction task (see below). Notice that `tpp` varies between the feature types. Hence, we subsequently use `tpp(F)` to denote the time per pixel for a specific feature type `F`.

After initialization, the agent observes the status of the service, and informs the coordinator about relevant changes. The status information includes the current load of the service (`Ls`), the load of the machine, and whether the service is available or not. The load of the service indicates how long the service is occupied with the tasks sent by the coordinator. The service maintains these tasks in a queue (cf. Figure 1 (b)) ordered by the priority of the task (see below) and the insertion time. Whenever a new task arrives, the agent estimates the cost to perform this task. The sum of theses costs is then the current load of the service (`Ls`). On the other hand, the load of the machine indicates how much work the machine has to do besides of the extraction of features. This load is expressed via the parameter `tpp`:

[1]The only problem occurred with large images where some machines begun to swap, and the cost grew more than linear.

If the load gets high, `tpp` becomes larger, if the load gets low, `tpp` becomes smaller. To determine this load, the agent simply measures the cost of an extraction task and compares it with the estimated cost. If the difference is large, the load of the machine changed and `tpp` is re-computed based on the times of the last couple of extraction tasks. In the database of each agent/service, there are two tables containing the above information:

TaskQueue

TaskDesc	Priority	InsDateTime	Est.Cost

FeatureCost

FeatureType	`tpp`	Avg.Cost

Information gathered and maintained by the coordinator: For each service type, the coordinator maintains a table containing the system parameters of the service, and some further parameters indicating the current status of the component, i.e. availability and load. Based on this information, the coordinator determines which service is best suited to perform an extraction task. Notice that due to the use of database tables, we dynamically can add and remove extraction services without restarting the coordinator. The component information is maintained in two database tables:

ExtractionService

CompID	Available	Load (`Ls`)	#Processors	Memory

ExtractsFeature

CompID	FeatureID	`tpp`	Avg.Cost

Apart of this component information, the coordinator stores status information about all current processes. In the context of preprocessing (cf. Figure 1 (b)), the coordinator maintains a queue, the so-called *extraction task queue*, that contains the extraction tasks of all processes that have not been handled yet. This queue is ordered by the priority of the task and the time of its insertion. The priority is assigned based on the process' type: Tasks of a SimilaritySearch-process have a high priority (H), all other tasks have a low priority (L). There are two different types of threads maintaining this queue: The first kind (TINS) inserts extraction tasks of new processes into the queue. The second kind (TPROC) pops the head task from the queue and processes it. Processing a task involves the following steps: 1) selecting a suitable feature extraction service (cf. next subsection), 2) sending the task to the selected service, and 3) keeping track of which tasks have been sent to which service (table *SentTasksList*). If the result of an extraction task arrives, the entry in the table *SentTasksList* is removed and the according process is continued as follows: If there are other tasks of the process not yet handled, the process must further wait, otherwise the next step of the process is executed. The following tables hold information about the preprocessing of current processes:

ExtractionTaskQueue

TaskDesc	InsDateTime	Priority	Delayed

SentTasksList

TaskDesc	CompID	SentDateTime

Peculiarities of the architecture: There are two special aspects in our architecture: Firstly, the CoMid maintains only meta-information about images and does not hold copies of the image data. When the coordinator requests a service to extract a feature from an image, it only sends information about where the image is stored, and how it can be accessed. Consequently, services and clients must fetch the raw image data directly from one of the specified repositories. Secondly, as a consequence of the first point, the coordinator has no information about the size of images not yet stored in the meta-database. Therefore, the coordinator cannot estimate the cost of an extraction task if the image is new. As we will see in the following, these peculiarities have no effect on the system's performance.

2.4 Service Selection

Each TPROC-thread (cf. Figure 1 (b)) pops the head task of the *extraction task queue*, selects a suitable service, and sends an extraction request to the chosen service. An extraction request consist of a feature type `F` and an image `I` that is described by the address of the repository holding the image and its local name in that repository. With the information described in the previous subsection, the TPROC-thread selects a service as follows:

1. Select all available services that can extract the specified feature `F`.
2. If the task has a low-priority, filter out services with a load `Ls` larger than `T`.
3. If more than one service remains, choose the ones with minimal `Ls`.

	1 FEE	9 FEEs
cheapest feature	5s	5s
most expensive feature	21s	21s
coordination of process	4s	4s
extraction of features	98s	21s
time to insert image	102s	25s

Table 1: Measured times for the insertion of an average-sized image (resolution 1024×796).

4. If more than one service remains, choose the ones with minimal `tpp(F)`.
5. If more than one service remains, choose one arbitrarily.

If no service is selected or available, the TPROC-thread re-inserts the task into the queue, and delays the execution of the task by a specified amount of time (e.g. 5 minutes). This delay is necessary to avoid that the next TPROC-thread handles the task again, since it is very likely that there is still no service available for that task. One can see this delay as an additional condition when popping a task from the queue. As such, we need to extend the table *ExtractionTaskQueue* with an additional column that stores this delay information (column Delayed), i.e. the point in time when the task can be handled again.

If selection is successful, the thread sends a request to the chosen service. Its agent notices this new task and estimates the cost for it. However, recall that a request does not contain the raw image data but only the information about where it is stored. Hence, the agent cannot estimate the cost based on the size of the image but must use an average cost for the extraction of feature `F`. Once the service has fetched the image, the agent can refine its cost estimation with the help of the formerly developed model. Further, the agent continuously computes the current load of the service (`Ls`) and informs the coordinator if the value has significantly changed. However, due to network delays, the load information of the selected service in the coordinator's database is not immediately updated. This can cause problems if many threads concurrently select a service for a task: Since the update of the load is delayed, they all may select the same service instead of different, equally suited services. To overcome this problem, we only allow one thread to select a service at any time. After the selection, the load of the service is increased such that the next thread will not choose the same service if equally suited services exist. As a result, subsequent tasks are distributed in a round robin fashion among the available services until the agents have updated the load in the coordinator's database. Notice, that this problem only occurs because the coordinator has no information about the size of the image. If the coordinator would know about the size of images, it could simply increase the load of the selected service by the estimated cost to extract the feature.

The idea behind the threshold `T` in step 2 is not to overload services with requests. Assume that we change step 2 such that services with `Ls` $\geq$ `T` pass through. Since the distribution of tasks is much faster than the actual evaluation, the coordinator would immediately assign all tasks of a Bulkload-process to the available services. The distribution is optimally according to the current information on the load of the machines. However, over the time, the load may vary and the distribution is no longer optimal. Even more, if we add a new service after the distribution but before the end of the process, it would not receive extraction tasks unless we re-distribute the tasks of another service. This threshold `T` is not considered in case of a high-priority task to enforce an immediate processing.

2.5 Failure Handling

If the coordinator notices the failure of a service or is informed about it, it has to re-distribute the tasks sent to this service. Otherwise the processes waiting for the completion of these tasks are blocked until the service recovers. Assume that service A fails, and that either the agent or the coordinator notices this event. Then the coordinator re-inserts all tasks of the *SentTasksList* belonging to service A into the *ExtractionTaskQueue*. Subsequently, these tasks are distributed among the remaining available services. If service A recovers, it has to delete its queue, and to wait for new tasks from the coordinator.

3 Evaluation of the Architecture and Experiments

In a preliminary experiment, we measured the time to insert an average-sized image (resolution 1024x796) with 9 feature types to extract in two different settings: a) only one extraction service (1 FEE), b) as many extraction services as there are feature types, i.e. 9 services (9 FEEs). Table 1 lists the figures of this experiment. Notice that the cost of extracting a feature varies from 5 to 21 seconds depending on the feature's type. Compared to the 4 seconds used for the coordination of the process[2], we see that the time to perform the InsertObject-process is dominated by the feature extraction effort. Even when employing a maximum number of extraction services, i.e. a service for each feature type, the cost of the preprocessing step is 5 times larger then the one of coordination.

In the following we consider the example of an upload of an image repository with N images. For each image, the system has to extract k features. Let T_{ij} denote the cost of extracting the j-th feature from the i-th image. Then the total cost for the preprocessing step is:

$$C(N,k) \;=\; \sum_{i=0}^{N-1}\sum_{j=0}^{k-1} T_{ij}$$

In our image retrieval system, we deployed a cluster of $m = 7$ PCs (WindowsNT) with an Oracle/8 instance on each of them. The image database consisted of $N = 100{,}000$ images gathered automatically from the web. The size of the images varied from small (100x100) to very large (8000x6000). The content of each images was described by $k = 9$ features. The total cost for the preprocessing of an upload of the entire database is: $C(N,k) = 113.4\ days$. In the following, we compute the minimum cost when employing $m > 1$ services, and the worst cost when employing our CoMid. The results are summarized in Table 2. Further, we describe the effects of a service failure and a search concurrent to a bulkload-process.

3.1 Cost of Optimal Distribution

With $m > 1$ components, a distribution of the extraction tasks is optimal if each component must extract features for the same amount of time. Consequently, the minimum cost for the preprocessing step of a bulkload-process is:

$$C^{opt}(N,k,m) \;=\; \frac{1}{m}\sum_{i=0}^{N-1}\sum_{j=0}^{k-1} T_{ij} \;=\; \frac{1}{m}C(N,k)$$

For instance, the optimal cost for a bulkload of our image database with $m = 7$ is $C^{opt}(N,k,m) = 16.2\ days$. However, the above formula does not hold if the number of images is small. For instance, if $N = 1$ and $m \geq k$, then the optimal cost is determined by the cost of the most expensive feature (InsertObject and UpdateObject process):

$$C^{opt}(1,k,m \geq k) \;=\; \max_{0 \leq j < k} T_{0j}$$

Notice, that in this case only k out of m services extract features, and that a further increase of m does not lead to a reduced preprocessing cost. The above formula also denotes the optimal cost of preprocessing a SimilaritySearch-process. In this case, k represents the number of features required by the similarity measure. In our image database (cf. Table 1), this means that a similarity search with a new image lasts at least 21 seconds if the most expensive feature type is required by the similarity measure.

3.2 Cost of Preprocessing with the CoMid

With a non-optimal distribution, it may happen that at the end of a bulkload-process some services are idle while others have still some tasks to process. With the service selection of subsection 2.4, we can guarantee that at any point in time no service has a load larger than `T`. Therefore, if a service gets idle after the distribution of the tasks of a bulkload-process, no other service must extract features for longer than `T`. Consequently, the cost of preprocessing a bulkload-process with our CoMid is limited by:

$$C^{opt}(N,k,m) \;\leq\; C^{CoMid}(N,k,m) \;\leq\; \mathtt{T} + C^{opt}(N,k,m)$$

[2]This includes the update of the meta-database and the external indexes, and the communication between the components.

Similarity search	cheapest feature		most expensive feature	
	optimal	our CoMid	optimal	our CoMid
only index lookup	1s	1s	30s	30s
only extraction	5s	5s	21s	21s
total effort for image from DB	1s	3s	30s	32s
total effort for new image	6s	8s	51s	53s

Insertion of N images with $k = 9$ features	1 FEE		7 FEEs	
	optimal	our CoMid	optimal	our CoMid
$N = 1$, only extraction	98s	98s	21s	21s
$N = 1$, total cost	98	102s	21s	25s
$N = 100,000$, only extraction	113.4 days	113.4 days	16.2 days	16.2 days (+10 minutes)
$N = 100,000$, total cost	113.4 days	113.4 days	16.2 days	16.2 days (+10 minutes)

Table 2: A summary of the measurements with our prototype system.

With the figures of our image database and a typical setting of $\mathtt{T} = 10$ *minutes*, the cost of the preprocessing with our CoMid is almost optimal, i.e. the difference to the optimal costs is only a few minutes compared to 16 days. As with optimal distribution, if $N = 1$ and $m \geq k$, the cost of preprocessing is also dominated by the cost of the most expensive feature:

$$C^{CoMid}(1, k, m \geq k) \;=\; C^{opt}(1, k, m \geq k) \;=\; \max_{0 \leq j < k} T_{0j}$$

This formula also describes the cost of extracting the k features for a SimilaritySearch-process if there are no other concurrent processes. In case of a concurrent process, e.g. a Bulkload-process, a service first must finish its current task before it can start with the high-priority task of the SimilaritySearch-process. As a result, the duration of a SimilaritySearch-process is lower-bound by the above formula. Let the superscript B and S denote values for the bulkload and the similarity search process, respectively. Then, the cost of preprocessing the SimilaritySearch-process is bound by:

$$C^{opt}(1, k^S, m \geq k^S) \;\leq\; C^{CoMid}_{SimSea}(1, k^S, m \geq k^S) \;\leq\; C^{opt}(1, k^S, m \geq k^S) + \max_{0 \leq i < N^B,\; 0 \leq j < k^B} T^B_{ij}$$

3.3 Dynamic Service Insertion and Removal

In a number of experiments, we have dynamically added an extraction service while uploading a larger portion of our image database. After the initialization phase (i.e. the computation of the system parameters), the coordinator immediately included the new service for the preprocessing. Since the load of the new service was 0 at the beginning, the next few tasks were sent to it until its load was about the same as in all other services. From there on, the system behaved as usual. On the other hand, after the removal of a service, the coordinator re-distributed its tasks and no longer sent tasks to it. The only problem in this case was the detection of the removal. For instance, a removal may occur due to a service or network failure. In most cases, the coordinator immediately detected the removal due to aborted network connections. However, in some cases the detection was not so fast (e.g. a component got disconnected from the network), but had no considerable influence on the overall performance. Only when a high-priority task of a search process was issued to a failed service, one could notice a delay.

Summary: Table 2 summarizes the measurements with our prototype system consisting of $m = 7$ feature services. The upper table lists the optimal and measured times for a similarity search with an image in the database and a new image as a function of the feature type (cheapest vs. most expensive). Furthermore, it depicts the cost of the index lookup (with the VA-File [15]) and the extraction in separation. The lower table lists the optimal and measured times for the insertion of a singe image, and the bulkload of the entire image database with 1 and 7 feature extraction services.

4 Discussion and Conclusion

In this paper, we have presented a coordination middleware for a highly-distributed, component-based architecture that near-optimally distributes extraction tasks of different processes among a set of available feature services. The flexibility of our approach allows to dynamically add and remove services without the need of restarting the system or any process. Furthermore, the deployed transactional process management overcomes service failures by initiating forward recovery. Our implementation demonstrates that replication at the service level and high-level parallelization successfully solves performance problems. We see this also as a confirmation that coordination layers can be built using database technology applied to meta-data about components and their status.

References

[1] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, 23–25 May 1990.

[2] S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. of the Int. Conference on Very Large Databases*, pages 28–39, 1996.

[3] K. Böhm, D. Ma, G. Nerjes, H.-J. Schek, and R. Weber. Metadata management with the hermes coordination middleware. Technical report, ESPRIT project HERMES (no. 9141), August 1998. Available at http://www-dbs.ethz.ch/~weber/paper/HERMESmeta.ps.

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of the Int. Conference on Very Large Databases*, Greece, 1997.

[5] A. Csillaghy. Retrieving information from digital solar radio spectrograms. In *Coronal Magnetic Energy Releases (Lecture Notes in Physics)*. Springer Verlag, 1995.

[6] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *Computer*, 28(9):23–32, Sept. 1995.

[7] G. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the Fourth International Symposium on Advances in Spatial Database Systems (SSD95)*, number 951 in Lecture Notes in Computer Science, pages 83–95, Portland, Maine, Aug. 1995. Springer Verlag.

[8] H. V. Jagadish. A retrieval technique for similar shapes. In J. Clifford and R. King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 208–217, Denver, Colorado, 29–31 May 1991. *SIGMOD Record* 20(2), June 1991.

[9] B. Manjunath and W. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, August 1996.

[10] H. Schuldt, G. Alonso, and H.-J. Schek. Concurrency Control and Recovery in Transactional Process Management. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'99)*, Philadelphia, Pennsylvania, USA, May 31-June 2 1999.

[11] H. Schuldt, H.-J. Schek, and M. Tresch. Coordination in CIM: Bringing Database Functionality to Application Systems. In *Proceedings of the 5th European Concurrent Engineering Conference*, Erlangen, Germany, April 1998.

[12] M. Stricker and A. Dimai. Color indexing with weak spatial constraints. In *Storage and Retrieval for Image and Video Databases IV, 2670 SPIE Proceedings Series*, 1996.

[13] M. Stricker and M. Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases, SPIE*, San Jose, CA, 1995.

[14] R. Weber. Parallel va-file. Technical Report 25, ESPRIT project HERMES (no. 9141), October 1997. Available at http://www-dbs.ethz.ch/~weber/paper/HTR25.ps.

[15] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. of the Int. Conference on Very Large Databases*, volume 24, New York, USA, August 1998.