Higher Order Databases and Multimedia Information*

Hans-Jörg Schek Roger Weber
Institute of Information Systems
ETH Zentrum, 8092 Zurich, Switzerland
E-mail: schek,weber@inf.ethz.ch

Abstract

This paper has two parts. In the first part, we give an overview of higher order databases, the research framework and vision of the database group at ETH Zurich. Higher order databases will be - or at least will contribute to - the infrastructure for distributed information systems engineering of the future. While a traditional database administers data, a higher order database administers components and specialized services. In particular such components can be databases. Then a higher order database is a coordinator of a database cluster. In general, a higher order database coordinates the components of the cluster together with components dedicated to specific tasks.

In the second part we apply higher order databases to multimedia information systems. We show how a networked image retrieval and management system can be built. Its components consist of image servers, feature extraction services and specialized indexing components allowing fast image similarity search. They are coordinated by a DBHO through transactional processes that realize similarity search, insertion and bulk insertion of images. The key features of our system are: scalability with respect to feature extraction services, parallelization of indexing components by replication and adaptation to dynamic changes of available feature extraction components.

Part I: Higher Order Databases

1. Introduction

The database group at ETH Zurich has been involved in the cooperation between databases and specialized application services since long time. "Cooperative Service Management for Open Systems" (COSMOS) is our research framework since 10 years. Meanwhile more and more researchers question the role of databases for the future distributed information systems engineering.

Carey, Hellerstein and Stonebraker (in their seminar opening talk "A sketch of Regres" at the University of Berkley[1]) observe that all current databases have been designed with the technology of twenty years ago. They state that due to a three-tier architecture, data are at the bottom and application code is away from data in the middle tier. They also state that databases are "bloated" by object-relational features, by stored procedures and triggers, and by warehouse features. They conclude that we should rethink everything.

Brodie in a keynote speech [11] states that "the database era nears its end" because of their inability to cope with the vast increases in data and transaction volumes. Heterogeneity hinders interoperability and accessibility. Architectural complexity is another issue in view of the many engine and repository types with ad-hoc solutions for warehousing and mining.

The VLDB Endowment in 1998 established a future directions group[2]. In their report 1999[3] the members agreed that the database technology has achieved a high standard in research and development but its future role in a globally distributed information network is less central. When modern databases have been introduced 20 years ago they have been considered as infrastructure and main platform for application development. Nowadays one observes that the role of databases has changed. When it comes to the management of components and application services, distributed client/middleware/server computing, application frameworks, enterprise resource planning systems like SAP, XML, e-commerce, the role of the DBMS is the one of a storage manager "only" in all these cases and the DBMS is far away from the applications.

To bring such topics into the fold of database confer-

*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*.

[1] Made available through DBWORLD 1999

[2] Members were Agraval, Brodie, Carey, Dayal, Gray, Ioannidis, Schek, Wang, Widom

[3] Partially made available in DBWORLD 1999

ences, the working group proposed to distinguish between the two main research directions:

(1) core database technology

(2) infrastructure for information system development

While the core technology (1) is well established and will be further developed worldwide by the international DB research and development community, the infrastructure direction (2) is under-developed and under-represented at conferences. In all examples given above, the database role of providing a storage manager, i.e. core DB technology remains central. However, client application development takes place on other platforms provided by middleware technologies. Or client applications are obtained by customizing pre-fabricated ERP systems. In conclusion it is stated that the focus of our community should turn to the investigation of how core technology can become more widespread and usable. Similar observations have been expressed in other documents such as [39, 34].

Taking all these observations together, database groups, while keeping competency in core database technology, start moving up to higher levels where workflow and distributed component and object management takes place. In doing so, database technology will contribute to the next generation platform for information system development. The work on higher order databases is the contribution of the ETHZ database group to this new vision.

2. Higher Order Databases

2.1. What is a Higher Order Database?

First rough definitions are: A Higher Order Database (DBHO) is a database over databases. A DBHO administers objects that are composed of objects and transactions that are composed of transactions. A DBHO is a "hyper-database", i.e. the primitives are again databases like hyper-matrices in numerical analysis are matrices whose elements are matrices. A DBHO administers distributed components in a networked environment and provides a kind of higher order "data independence" .

The latter aspect deserves more explanations. Remember that data independence was a major breakthrough when relational databases were propagated. Now we must strive for immunity of application programs not only against changes in storage and access structure but also - and this is the point here - against changes in implementation and potential replication of complete components and their services. Our infrastructure should find the right alternate component or alternate evaluations even in case of unavailability of some. This looks similar to query optimization in a traditional DB. In a DBHO if, for performance reasons, components are replicated it is up to a DBHO to dynamically include them in the executions. We could even think one step further: Our system should replicate some "hot" ones automatically, once we provide more resources.

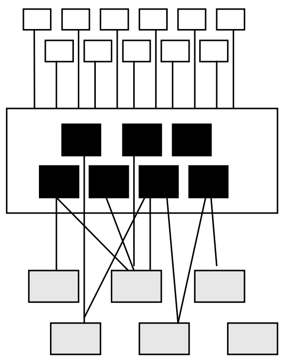


Figure 1. DBHO as a three-tier architecture

What is the difference between a DB and a DBHO? In a nutshell we say: A DB is a platform for clients concurrently accessing shared data. We need data definition, data manipulation, transactions at the interface. The DB "under the cover" performs query optimization, correctness for parallel access, recovery, persistency, load balancing, admission control, availability.

Similarly, A DBHO is a platform for clients, concurrently accessing shared application services. As opposed to shared data in a DB, in a DBHO we have shared components and services. At the interface of a DBHO we need component and service definition and description, service customization, transactional processes encompassing multiple service invocations. The DBHO, under the cover, performs optimization of client requests, routing, scheduling, parallelization, correctness of concurrent accesses, flexible failure treatment, providing guaranteed termination, availability, flexible recovery, scalability.

2.2. DBHO as middleware

In its most simple form a DBHO is a three-tier architecture as shown in Figure 1. Clients (white boxes) access application services (black boxes). The DBHO is a coordinator (circumscribing rectangle) managing concurrent processes consisting of several application service invocations. It further routes the invocations to the appropriate data servers and specialized components (grey boxes) that execute concurrent data access and execute specialized methods.

At this point we ask what the differences between a DBHO and other middleware architectures or systems are. It is clear that TP-monitors, enterprise resource planning systems such as SAP, extensible databases, federated databases and multi-databases, data warehouses, or the Corba infrastructure look similar to the DBHO architecture (cf. Figure 1). Each of them provides the one or the other desirable functionality of DBHO. But in reality, today, these systems are treated as different fields and developed by people from different disciplines. We aim at a unifying, more systematic architecture, once we have understood the essential principles. It will turn out that often database technology, when applied at a higher level of abstraction, higher than records or tuples, leads to a clearer understanding and to convenient implementations. This is the objective of the DBHO project. In the following we discuss some examples of existing middleware components and explain what a DBHO will add.

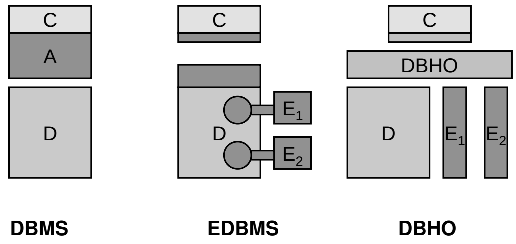


Figure 2. Evolution of extensible databases (EDBMS) to a DBHO architecture

2.2.1. TP Monitors

A TP-Monitor is, in a narrow definition, an operating system for transaction processing. More generally, it is infrastructure for developing and running applications, services, and components in a three-tier architecture [7]. A DBHO would add programming design tools for the specification of distributed applications and for an automatic generation of program code. Extended workflow systems are moving into this direction [4, 50].

Further a DBHO would add a real transaction layer above DB-transactions and transactional process management with failure treatment, availability, guaranteed termination and ”semantic” correctness. In current tp monitors all components are forced to the 2PC protocol, i.e. the coordination of all subtransactions is performed at one level of abstraction only, i.e. the data level. Finally a DBHO would add optimal routing to components and self-tuning facilities or configuration optimization by ”semantic” replication of whole components.

2.2.2. Enterprise Resource Planning (ERP)

As a next example we consider the architecture of the SAP system, a good representative of ERP software [32]. Clearly we observe the traditional DBMS at the storage level. All application code is at the middle tier in so-called application servers and presentation stays at the utmost level. The architecture is the one of a DBHO. However a closer inspection of the role of the middle layer and of the mapping to the DB layer in a SAP system makes clear that a real DBHO layer would add the following:

First and most importantly a real transaction layer on application objects would be added. It would provide transactional process management, i.e. semantic recovery and semantic correctness as well as the possibility of alternative executions in case of failures and guaranteed termination. In the current system there is a separate transaction layer on top of DB transactions called logical unit of work which is in the sense of a DBHO. However application programmer must aquire locks and so they are responsible for correctness in case of parallel access and in case of failures. Also consistency of application server buffers in case of updates is not provided automatically.

Second, a DBHO would add a flexible mapping to many database storage managers. In the current system there is one big database only, hindering scalability. If many more than one storage component would be allowed, optimal decomposition and routing of client requests to storage managers can take place and an automatic replication of whole storage components based on usage patterns can be achieved.

2.2.3. Extensible Databases

Extensible databases are not considered as middleware architectures but they add application specific code to the core database system providing clients a two tier architecture [29, 40, 26]. Insofar they must be considered as a direction that has bridged the gap between the applications and the core database. However the disadvantage is that a higher level of abstraction is artificially moved into the low level of a database. It is a clearer and more flexible architecture to coordinate several components providing specialized data types and computational methods together with general purpose databases. The evolution from extensible databases to a DBHO architecture is shown in Figure 2. The direction of datablades (Informix [20]), cartridges (Oracle [28]) or extenders (IBM [19]) is considered a good step in principle however, as said, they contribute to the ”bloated” complex database in a two-tier architecture as opposed to a three-tier DBHO architecture.

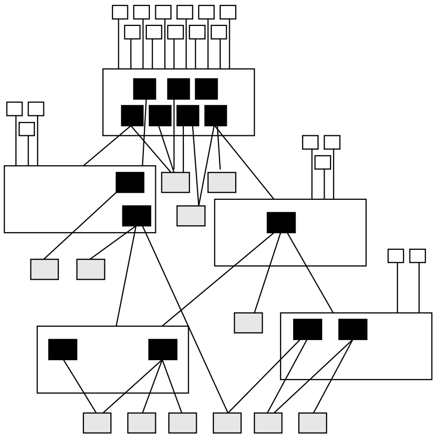


Figure 3. Complex DBHO architecture, several coordinators work together

2.3. DBHO Research

2.3.1. DBHO Complex "Big Picture"

The general architecture where DBHO concepts are used is shown in Figure 3. Clients (white boxes) invoke application services (black boxes) on server components that are coordinated (circumscribing rectangle). One application invocation in turn may use several other coordinated components and invoke one or several lower-level application services there. This continues along the invocation hierarchy until we finally reach leaf nodes. The leaf nodes perform data calls or specialized computations (grey boxes). Note that coordinator nodes play the typical middleware double role: they are servers for clients and clients to other servers. The repeated use of coordinators adds complexity because at any server node we may have additional clients also accessing our distributed system concurrently to others coming from higher level coordinators. It becomes clear that this is a generalization of federated databases where we distinguish local and global objects as well as local versus global transactions [15, 24]. Research in this area will be highly relevant to DBHO.

In view of the previous discussion it is already obvious that many questions deserve foundational and experimental research. We will summarize some aspects in the following and then concentrate on multimedia components in Part II as a detailed case study for DBHO research.

2.3.2. Composite Transactions

We have studied the problem of ensuring correctness of concurrent executions in architectures like the one in Figure 3. A coordinated server component, called coordinator in the following, contains a full transaction manager for the client transactions encompassing invocations of several application services. Every coordinator in the composite system performs its transaction management ensuring (local) correctness and (local) recovery. The problem is how global correctness and global recovery is ensured, given that each coordinator locally works correct. In other words we must know what additional ordering restrictions a "caller" coordinator must impose and hand over to a "called" coordinator. What handshaking between coordinators is necessary in order to correctly control concurrency in composite systems as shown in Figure 1? We have extensively studied this problem in the past from a foundational point of view [48, 36, 49, 1, 2], we have studied practical protocols [35, 33, 18] and performed several evaluations [23, 31].

2.3.3. Transactional Process Management

In transactional processes we go beyond transactions in that we not only specify conflicts between invocations but we also know about compensation and about retrieability. Moreover, in a transactional processes we allow to specify alternate executions and, based on these, we generalize atomicity of transactions to a notion called guaranteed termination. It means that a single process will eventually terminate along a well defined path even in case of failure. Abort is a special option, forward recovery another. The guaranteed termination will also be ensured if several transactional processes run concurrently. This work has been documented, e.g. in [37]. Work on transactions and transactional processes is a foundation and a basis for of DBHO transaction implementations.

2.3.4. PowerDB

In the PowerDB project we explore a DBHO consisting of a homogeneous set of component databases in a PC cluster [17, 30]. In every component we have a complete DBMS with its data. Clients access data via the coordinator, i.e. via the DBHO. We explore protocols for high-level transaction management under special consideration of replication. Replication of complete databases contributes to considerable speed-ups in case of read transactions. Due to the second layer transaction management we avoid a 2PC and avoid synchronous updates. In addition query routing aims at detecting components that have the shortest response time due to queries that have been processed before. Replication can be full or be restricted to certain parts of the database. We investigate methods that dynamically allow to add more components to the cluster. Similar objectives are pursued by the group around Kitsuregawa [43].

Part II: Multimedia Information

In this part we show how the vision of Higher Order Databases is realized in the context of multimedia information systems. In order to provide "openness", flexibility, and scalability, we let a number of specialized components work together, and have a DBHO as coordinator. The (low-level) services provided by the (external) components are combined to higher order services offered to clients. Compared to Figure 3, we only have one coordinator, i.e. a three-tier architecture as shown in Figure 1. We report on the architecture of such a system, on the development of components, and on the implementation of the coordinator. Furthermore, we describe two scenarios where the parallelism inherently possible by such a system leads to a significant speedup in performing higher order services.

The various aspects of this prototype system are presented in more detail in a number of publications [46, 45, 4, 37, 9, 44, 47]. In the following, we give a coherent, compact presentation of the whole system.

3. Design Considerations of a Distributed, component-based architecture

This section introduces the architecture of our image retrieval system. We justify our design decision by considering the requirements of autonomy, extensibility, and performance.

3.1. Autonomy

Consider an image provider who wants to charge customers for accessing its high-quality images. In order to keep autonomy and to control the distribution of the contents, such a provider would like to export only the metadata of the images to the retrieval system to limit the distribution of images to customers who pay for the items. Via a DBHO coordinator, customers need not search for items at many places, but they can use a central service to satisfy their information needs. Coordination of the repositories is required which enables the meta-database to keep track of the status of a repository, that is, a provider must allow observation of repository changes that are relevant to the meta-database. In our approach, so-called coordination agents are plugged into the repositories. Whenever the contents of a repository is changed by the provider, the corresponding agent triggers an update of the meta-database.

3.2. Extensibility by Specialized Components

Content-based image retrieval relies on the extraction of specific features from the images, such as color [42, 41, 14], texture [27], or shape [21, 25] information. The quality of the retrieval primarily depends on the features' ability to describe the contents of the images. Since content-based image retrieval is a rapidly changing research area, the system must be extensible with respect to new feature extraction methods. While the features are crucial for the assessment of similarity between two images, indexes are crucial for the performance of the retrieval. Hence, index structures like X-tree [6], M-tree [12], or VA-File [47] are developed as specialized components. To ensure extensibility, feature extraction, index maintenance of feature values are external components coordinated by the DBHO architecture. As with data repositories, coordination agents observe these components to provide status information such as availability and current load to the meta-database system.

3.3. Coordination

Coordination is required to correctly react to changes in the contents of image repositories and to maintain consistency. Coordination between the search-engine and all the external components is also required to avoid or mitigate problems incurred by unavailable components. Furthermore, coordination (between the image servers and the search-engine) is beneficial to distribute requests among the available components to reduce the overall cost, and to direct clients towards image servers that allow for fast image delivery. These tasks require additional information about the components such as availability, load, and bandwidth information which are gathered by the coordination agents plugged-into all external components. Figure 4 depicts the component-based architecture of our image retrieval system. The central system consists of the meta-database, of a simple search engine, and of the coordinator. The coordinator is assisted by coordination agents that are plugged into each external component and that inform the coordinator about status and contents changes. The remaining components store the image data (image server), process meta-information (feature service, index service), or assist users to retrieve items of interest (client).

3.4. Replication and Parallelization

A serious problem of any image search and retrieval system is the cost of extracting features, and of searching for similar images. Feature extraction is necessary when new images are inserted or when a similarity search with a new image must be supported. It is obvious that due to this expensive preprocessing, it looks more promising to utilize a large number of components to preprocess the images and to determine the various feature representations. The task of the coordination middleware is to distribute tasks among the available components such that each component is equally loaded. A similar problem appears in the context of *nearest neighbor searches* (NN-search) in the feature indexes when

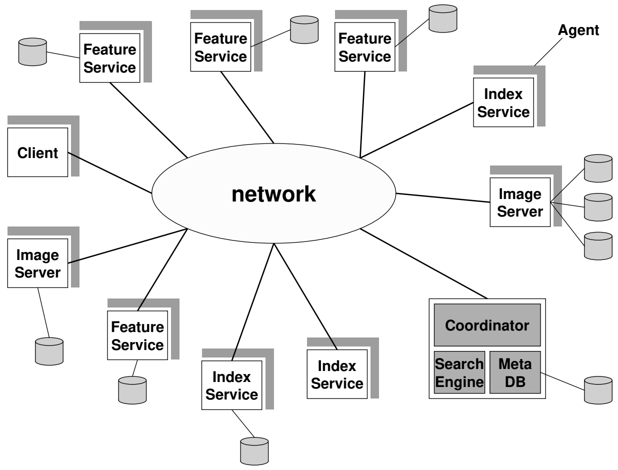


Figure 4. DBHO for multimedia components

feature type	#dim.	extr. cost [s]
color moments, 1 region	9	7.3
color moments, 5 regions	45	13.6
color moments, 35 regions	315	21.1
color histogram, 1 region	64	4.9
color histogram, 5 regions	320	9.2
color histogram, 35 regions	2240	16.5
texture moments, 1 region	30	5.9
texture moments, 5 regions	150	6.9
texture moments, 35 regions	1050	11.0

Table 1. Feature extraction costs for an image of size 1024x796.

performing a similarity search. Due to the high cost of NN-searches, it also looks much more promising to employ a large number of components to search the indexes in parallel instead of using only a single component. Whenever we detect a bottleneck in the system, be it feature extraction, be it indexing of feature spaces, or be it data delivery, we replicate that services and dynamically add new components to the system. After their registration, the coordinator will account for these new components and route requests to them without the need of reconfiguring or restarting the system. The same holds true if components retire from the system or fail: the coordination agents notify the coordinator about this, and the coordinator will exclude that component from future processing, and has to re-route pending tasks sent to that component.

4. Specialized Components

This section describes the main components, i.e. image servers, feature extraction services, index services, and the coordinator. We also explain the VA-File approach that supports similarity search in high-dimensional vector spaces.

4.1. Image Servers

The main purpose of an image server is to provide clients with image items. In our prototype system, we have integrated two different kinds of image servers: (a) a simple server, and (b) a network aware server [10]. While the first kind of server simply delivers the requested images, the second kind adapts the quality (and thus the amount of data) to the available bandwidth of the network path from the server to the client. Each image provider must allow for the integration of a coordination *agent*. The agent's task is to watch the contents of the repository, and to identify any relevant changes like insertion of new images.

4.2. Feature Extraction Services

A feature extraction service facilitates the extraction of features from image data. Any component may send an image (or the address of the image, e.g. its URL) to the feature extractor and retrieves a set of features representing the image. Typical feature types for images are color [42, 41, 14], texture [27], and shape [21, 25] information. For example, the texture descriptors specified by Manjunath and Ma [27] 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. Orthogonally to these core feature types, one can partition the image in a number of (overlapping) regions. For each region, one computes the features, combines them to an overall feature representation of the image. This enhances the core feature types with spatial information. In our prototype system, we have implemented a number of these feature extraction algorithm. Table 1 enumerates them together with their dimensionalities, and the extraction cost for an average sized image. Note that the costs are rather high, i.e. between 5 and 21 seconds for an average sized image.

4.3. Index Service

An index service maintains the features of a collection of images, and performs similarity searches on behalf of the coordinator. When a user requests for similar images, the coordinator finally routes the query to one of the index services available. Then, the coordinator adds meta-information to the answer of the index service, and returns

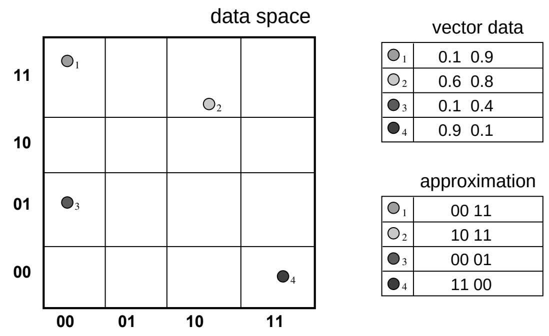


Figure 5. Building the VA-File

this result to the client. A common implementation of similarity search is nearest neighbor search (NN-search). As shown in Table 1, the dimensionality of features can be rather high. A number of theoretical and experimental work has shown that indexing such high-dimensional spaces often results in linear search complexity [6, 5, 22, 47, 8]. In consequence, it is much more efficient to utilize linear structures instead of tree-based structures. Our approach to this is called *vector approximation file* (VA-File), a linear approach using approximations of the vectors. All the index services in our system are based on the VA-File. Due to the linear structure of the VA-File, we are able to utilize several index services in parallel to speedup similarity search. In the following, we shortly overview how the VA-File works. Parallelization of the VA-File is presented later on.

4.4. A Short Overview of the VA-File

The *vector approximation file* (VA-File) divides the data space into 2^b rectangular cells where b denotes a user specified number of bits (e.g. some number of bits per dimension). Instead of hierarchically organizing these cells like in grid-files or R-trees, the VA-File allocates a unique bit-string of length b for each cell, and approximates data points that fall into a cell by that bit-string. The VA-File itself is simply an array of these compact, geometric approximations. Nearest neighbor queries are performed by *scanning* the entire approximation file, and by excluding the vast majority of vectors from the search (filtering step) based only on these approximations.

4.4.1. Compressing Vector Data

For each dimension i, a small number of bits (b_i) is assigned (b_i is typically between 4 and 6), and 2^{b_i} slices along the dimension i are determined in such a way that all slices are equally full. These slices are numbered $0, \ldots, 2^{b_i} - 1$ and are kept constant while inserting, deleting and updating data points. Let b be the sum of all b_i, i.e. $b = \sum_{i=1}^{d} b_i$. Then, the data space is divided into 2^b hyper-rectangular cells, each of which can be represented by a unique bit-string of length b. Each data point is approximated by the bit-string of the cell into which it falls. From another perspective, we use a quantization scheme to derive approximations from the vector data.

Figure 5 illustrates this for five sample points. In addition to the basic vector data and the approximations, only the boundary points along each dimension must be stored. Depending upon the accuracy of the data points and the number of bits chosen, the approximation file is 4 to 8 times smaller than the vector file. Thus, storage overhead ratio is very small, on the order of 0.125 to 0.25.

4.4.2. The Filtering Step

When searching for the nearest neighbor, the entire approximation file is scanned and upper and lower bounds on the distance to the query can easily be determined based on the rectangular cell represented by the approximation. Assume δ is the smallest upper bound found so far. If an approximation is encountered such that its lower bound exceeds δ, the corresponding object can be eliminated since at least one better candidate exists. Analogously, we can define a filtering step when the k nearest neighbor must be retrieved.

4.4.3. Accessing the Vectors

After the filtering step, a small set of candidates remain. These candidates are then visited in increasing order of their lower bound on the distance to the query point Q, and the accurate distance to Q is determined. However, not all candidates must be accessed. Rather, if a lower bound is encountered that exceeds the (k-th) nearest distance seen so far, the VA-file method stops. The resulting number of accesses to the vector file is very small. For example, consider a 10th nearest neighbor searches in a 50-dimensional, uniformly distributed data set ($b_i = 6$). At $N = 50'000$, only 19 vectors are visited, while at $N = 500'000$ only 20 vectors are accessed. Hence, apart of the answer set (10) only a small number of additional vectors are visited (9–10).

4.5. Coordinator

The coordinator is the key component of a DBHO. In our system, it maintains the consistency between data repository and meta-database, it performs processes to update or query the image database and guarantees their correctness, it gathers and maintains information about the components such as their current load or their status, and it balances the load of a set of identical services (e.g. a set of feature extraction services). The coordination agents in the external components assist the coordinator, and provide it with status and contents information about the components. In the

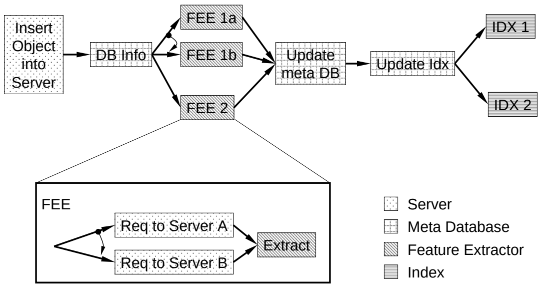


Figure 6. Execution graph for the InsertObject **process**

following section we will describe the coordination of important processes in detail.

5. Coordinating Higher-Order Processes

In our distributed image system, examples of higher-order processes or services are inserting a new image or searching for similar images to a given reference image. The coordinator executes these processes based on their description and ensures transactional guarantees on the higher, semantic level. As we have discussed above (Section 2.3), we apply *transactional process management* [38, 37].

5.1. Examples of Transactional Processes

In the following, we will sketch the relevant steps of three transactional processes to which we will refer later on.

Process InsertObject (cf. Figure 6): Assume that an image provider has inserted new items into a repository. The corresponding agent of the repository notices the insert operation and triggers the start of the InsertObject-process. The first step is to gather meta-information about the image, e.g., owner, size, name, resolution, and storage format. This information is then inserted into the meta-database, i.e., a new image object is created in the meta-database, and the related meta-information is inserted. To enable content-based retrieval, feature extraction engines must determine the feature values for the new image. They request the image from the server, determine the feature values, and insert these values into the meta-database. Furthermore, all the external indexes must be updated that correspond to the features extracted.

Process Bulkload: Assume that the system is extended by a new repository. In this case, we need to insert a large number of images concurrently. Thus, a Bulkload-process can be seen as a process consisting of a large number of InsertObject-subprocesses executed concurrently. However, the primary goal is not to accelerate a single insertion of an object, but to accelerate the insertion of all images together. For example, preprocessing a single image should employ as many feature extraction components in parallel as possible to minimize the duration of the process. With a Bulkload-process, on the other hand, only one component should extract features for one image in order to reduce network costs and to reduce the load of the server holding the new repository.

With respect to Bulkload-processes, the coordinator must assign images to all available components in such a way that the overall duration of the process is minimized. Moreover, the coordinator should immediately employ components added to the system. In case of failures or removals of components, the process must terminate even if not all tasks sent to that component have finished. E.g. the coordinator must re-assign tasks of failed or removed components to other available components.

Process SimilaritySearch: Consider a user who issues a similarity search with a reference image stored on the client's machine. Since the reference image is not part of the retrieval system, features for that image do not exists in the meta-database. Hence, the first step is to extract the features required for the index lookup. Subsequently, the search engine accesses several external indexes (in parallel) to determine the answer set for the given query. Since the images are not stored in the meta-database, the client obtains only a list with image identifiers, ranks, relevance scores, and meta-information on each image. The meta-information includes placement information, i.e., a list of servers that store versions of the images. The client finally must fetch the images from the specified servers and present them to the user.

The search for similar images is a highly interactive process that requires the results of a query to be delivered "'reasonably fast"' to be useful for an end-user. In "'conventional"' search engines supporting only textual information the response time is dominated by the costs for searching/indexing the database. This is not the case for image search and retrieval systems where the feature extraction of the query image, as well as the index lookup (NN-search) can consume a substantial (often unpredictable) amount of time. Therefore, our system exploits parallelization by replication of important components.

Notice, that there are further processes in the system such as UpdateImage, DeleteImage, BrowseCollection, which exhibit basically the same characteristics as the two processes described above.

5.2. Transactional Process Management

As introduced in [38, 37, 3], a transactional process is a directed graph whose nodes, the steps, are conventional transactions. The component together with its agent therefore must provide atomicity and isolation at the level of single steps while the transactional process provides semantic atomicity and semantic isolation at the coordinator level. In addition, a transactional process provides the specification of alternative executions and distinguishes between restartable and compensatable steps. Semantic atomicity in this more general setting means the guarantee for a termination using the regular steps or one of the alternatives. Semantic isolation means that the coordinator is given the information about commutativity of steps.

Figure 6 depicts the execution graph of the InsertObject process. The first step, the insertion of a new image, is obviously not compensatable (a provider will not remove a new image). Hence, the remaining steps of the process must be restartable. For example, consider step FEE 1a. If the feature extraction fails, we may restart the step FEE 1a or we may use the alternative FEE 1b (depicted with an small arrow between the two large arrows). These alternatives illustrate the fact that there exist two (identical) feature engines that are able to extract the first feature. On the other hand, FEE 2 depicts that only one feature extraction engine can compute the second feature. Further, note that the extraction of features, e.g. FEE 2, can be seen as a sub-process of the InsertObject process consisting of a request step (with possible alternatives) and an extraction step. The update of the meta-database and the subsequent update of the external indexes is performed as soon as actions of all paths leading to Update meta DB are executed, i.e., as soon as all features are extracted.

Figure 7 illustrates a transactional process for the InsertObject process. The obvious difference to a distributed transaction is that the steps 1–6 are encapsulated by subtransactions. Each of these steps is repeatable. For example, if an index service fails, only step 6 is repeated but not the entire process as in the case of a distributed transaction.

In our system, we have implemented a coordinator that follows the above process execution graph by enqueueing steps of (concurrent) processes that must be executed next. The coordinator continuously dequeues a step, tries to execute it, and if successful, enqueues the subsequent step(s). If the execution is not successful, the coordinator tries to perform an alternative step. If none of the alternatives succeed, the step is enqueued again and executed later on. For example, consider the step DB Info in Figure 6. After having finished this step, the coordinator inserts two new steps into the queue (FEE 1a, FEE 2). Subsequently, the coordinator dequeues the next step (e.g. FEE 1a) and tries to execute it. If the step fails (e.g. the corresponding engine is not available), the coordinator tries to execute its alternative step (FEE 1b). If one of the alternatives is successful, the next step (Update meta DB) is inserted only if all other incoming paths of that step are finished as well. If all alternatives fail, the step is enqueued again and executed later on. Once all steps of a process are executed, the coordinator makes the new image visible to other processes (e.g., to search processes). To improve the throughput of step executions, the coordinator can execute several steps in parallel as described below. Further details of the realization of the coordinator and of its meta-database using database technology (transactions, triggers, stored procedures) can be found in [9].

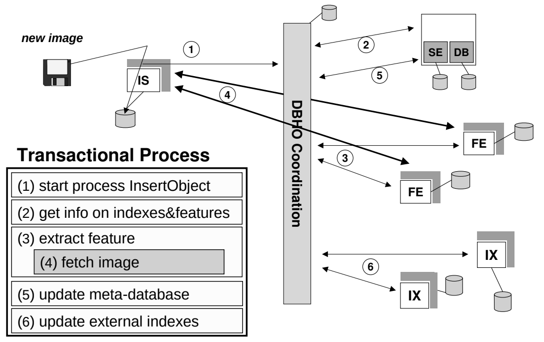


Figure 7. Executing InsertObject **in a transactional process.**

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

6. Parallelization and Replication

Content-based retrieval in image databases implies two very expensive actions:

1. preprocessing of images, i.e. extracting features from images, and
2. nearest neighbor similarity search

Referring to Table 2, extracting features from an image lasts over 100 seconds on average with the feature extraction methods in our system. Consider now a Bulkload-process with 100,000 images to insert. The overall preprocessing costs is more than 100 days. Thus, it is very natural to employ a large number of components to extract feature from images. Analogously, performing a NN-search to find similar images is very costly (cf. Table 3). Even if no features must be extracted (i.e. the reference image comes from the database), it takes more than 30 seconds to answer a simple query. As with features, it looks more promising to utilize several components in parallel to reduce the search costs significantly.

In the following, we look at parallelization of feature extraction tasks, and at parallelization of similarity searches. In the first case, the goal is to equally distribute the extraction tasks among the available components. In the second case, the aim is to divide a similarity search task into a number of sub-tasks, to let the components evaluate these subtasks, and to have the coordinator producing the overall result.

6.1. Ressource Management 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), 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 coordinator 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-process is minimal,
3. the tasks of a Bulkload-process optimally exploit the available services,
4. the influence of a service failure is minimal, i.e. no process is blocked, and
5. new resources can be added dynamically.

6.1.1. Necessary Meta-Information for the Preprocessing Step

Information maintained by agents: The agent of a feature extraction service is responsible to inform the DBHO coordinator about relevant changes of the load of the service and the machine. For each extraction task in the queue of the service, the agent estimates its cost based on a linear model. Let tpp denote the average time per pixel spent for the extraction of a specific feature. Thus, the cost for an extraction task is determined by: $cost = num.ofpixels \times \texttt{tpp}$.

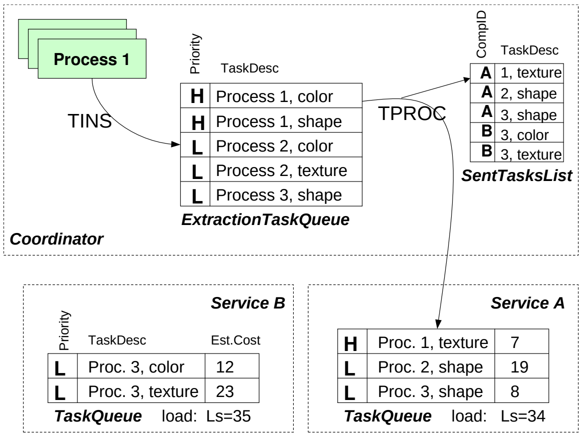


Figure 8. Maintenance of the extraction task queue.

Furthermore, the agent maintains average costs estimates for each feature type. These values are important for the DBHO coordinator in order to have a first, rough guess on how costly the extraction of an image is (see below). The information maintained by the agents is stored in two tables:

TaskQueue

TaskDesc	Priority	InsDateTime	Est.Cost

FeatureCost

FeatureType	tpp	Avg.Cost

Information maintained by the coordinator: 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 system ressources, and about how costly the extraction of the provided features is (i.e. parameters tpp and Avg.Cost from above). After initialization, the agent informs the coordinator about changes in the load of the service (Ls), and whether the service is available or not. The load of the service is measured as the time the service is occupied with its tasks. This information is stored in two 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 8), 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

6.1.2. Service Selection

Each TPROC-thread (cf. Figure 8 (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, discard services with a load `Ls` larger than `T`.
3. If more than one service remains, choose the ones with minimal `Ls`.
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). 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, a request does not contain the raw image data but only the information about where it is stored. Hence, the agent can only estimate the cost based on the 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. Furthermore, the agent continuously computes the current load of the service (`Ls`) and informs the coordinator if the value has significantly changed. As a consequence, if the coordinator requests a service to extract a feature, the new load of that service is not immediately updated in the coordinator's database. 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, the coordinator roughly estimates the duration of the extraction task with the average cost parameter of `F` for the choosen service. With this estimated duration, it can update the load of the service.

The idea behind the threshold `T` in step 2 is not to overload services with requests. Assume that we change step 2 such that all available services are considered. Since the distribution of tasks is much faster than the actual evaluation, the coordinator would immediately at the beginning assign all tasks of a Bulkload-process to the available services. The distribution is optimal according to the current information about 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, it would not receive extraction tasks unless we re-distribute the tasks of another service. A good choice of `T` is 10 minutes: In this case, we can guarantee that all services will finish their work for the Bulkload-process within an interval of 10 minutes. Note that `T` is not considered in case of a high-priority task to enforce an immediate processing.

6.1.3. 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 reinserts 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 the coordinator to receive new tasks.

6.2. Parallelization of NN-Search

An index service is a further example of a specialized component: it maintains indexes over feature spaces and performs similarity searches on behalf of the coordinator. As we discussed before, we use the VA-File, i.e. a linear approach to find the nearest neighbor. This means that response times increase linearly with the number of objects

Similarity search	smallest feature size	cost	largest feature size	cost
index lookup:				
1 service	5 MB	0.5s	1 GB	30s
3 services	5 MB	0.2s	1 GB	1s
cost of extraction		5s		21s
total effort for image from DB:				
with 1 service		0.5s		30s
with 3 services		0.2s		1
total effort for new image:				
with 1 service		5.5s		51s
with 3 services		5.2s		22s

Table 3. Cost of searching for similar images.

and the number of dimensions. Table 3 lists the search costs for some of the indexes in our system (120,000 images, features are listed in Table 1). Notice that for the feature type with the most dimensions, we have about 1 GB of feature data. Even with the VA-File, a search lasts more than 30 seconds. Clearly, this is not satisfactory if we consider interactive search scenarios. To speedup similarity search, our approach is to utilize a number of components in parallel. Table 3 also lists the response times of a parallel NN-search with three components. As one can see, search costs is significantly reduced: instead of 30 seconds with one component, it takes only 1 second with three components. This drastic reduction of processing time is due to main memory usage as opposed to disk IO as we will see below. The bottleneck of similarity search is now extracting the features from the reference images. Fortunately, this is only necessary for the first evaluation. All subsequent queries with the same reference image can reuse the earlier determined features.

6.2.1. Design Considerations for Parallel Index Search

Replication of the VA-File: Basically, there are two alternatives: We can partition a feature index over a set of components, or we can replicate the the entire index for each component. With the first alternative, to answer a similarity search, each component searches only its part of the index, and the coordinator combines the answers to the overall answer of the query. However, this imposes a number of problems: What can the coordinator do if one of the components fails? How can we add a new indexing component, and how to re-distribute the index data in this case. How can we take different system characteristics of the components into account (not all components are equally fast)? To positively answer these questions, our approach is to replicate the entire VA-File. In this case, all the above problems can be solved in a straight-forward way.

Replication of the Vectors: With the VA-File, we can replicate either both the vector and approximation data, or we only replicate the approximation data. With the first alternative, storage consumption is much higher. With the latter alternative, if the component with the only vector file fails, no similarity search can be answered any more, i.e. availability is worse. In our system, we always replicated the entire approximation and vector file. Thus each component is able to answer a query on its own and availability is maximal. We accept increased update cost and keep in mind that in update/insert processes each component is updated in parallel but asynchronously.

Heterogenous Environments: As with feature extraction before, we also must account for heterogenous components when evaluating similarity queries in parallel. Obviously, not all the components are equally fast and their performance may even vary over time (e.g. in case that other services run on the components). Figure 9 (a) depicts the evaluation of a query in a heterogenous environment in the case that we do not account for heterogeneity, i.e. if we send equally large sub-tasks to the components. The interval within the bars denotes the range in the files the component has to search through (e.g. in tens of thousands). In this example, components A, C and D are twice as fast as component B, and, thus, the overall search cost (14) is dominated by B. Furthermore, during the last 7 time units, only component B is doing work related to the query. On the other hand, if we adapt the size of the sub-tasks according to the (current) search performance of the component, we may significantly decrease search costs. In Figure 9 (b), for example, the coordinator requests component B to search only through half a large range in the files as the other components. As a result, the overall search cost is only 8 as opposed to 14.

Assignment of Sub-Tasks: Usually, the coordinator has the freedom to send a sub-task to an arbitrary component. However, it might be beneficial to restrict this freedom. For example, Figure 9 (b) shows the distribution of sub-tasks among the four available components. In this example, the only restriction we had so far is that B should only search half as much of the data as the other components. But we still may choose which range of the data we assign to B. For the query depicted in the figure, we assigned the range $[8, 12[$ to B. For the next query, we could assign the range $[0, 4[$ to B. However, it is better to let B again search through the range $[8, 12[$ because this part of the file may still be in the file cache of component B. As a consequence, no I/O operations are necessary to answer the new query and evaluation is hence much faster. This applies also to all other components. As a consequence, sub-

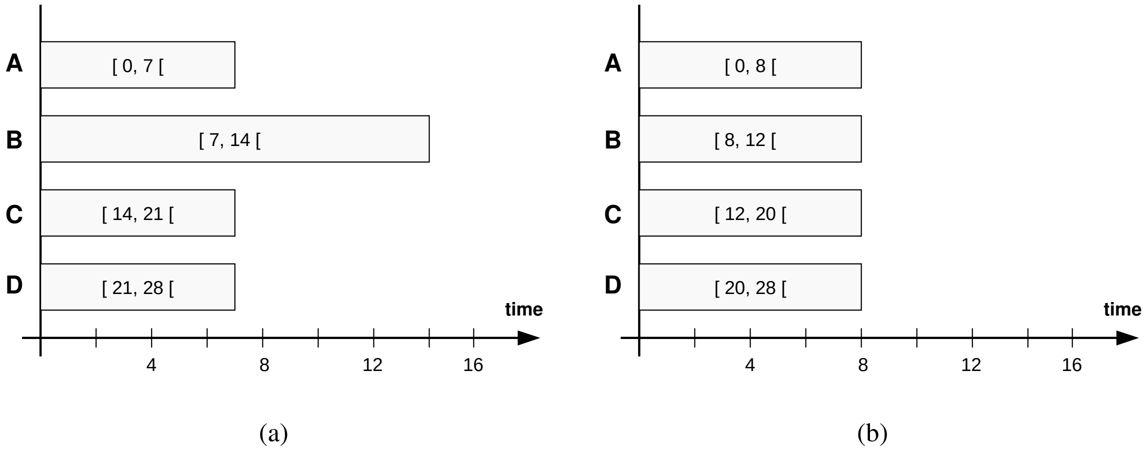


Figure 9. Query evaluation in a heterogenous environment (B is half as fast as the other components); (a) with equal partition sizes, (b) with dynamic partition sizes.

sequent queries are performed in the main memory of the components. From this we draw, that it is more promising to always send (almost) identical sub-tasks to the same component to benefit from caching effects.

7. Implementation Status and Evaluation

In the past two years, we have built a networked image retrieval system in a joint effort of database, network, and imaging researchers. The system consists of 7 kinds of components that are glued together by a DBHO.

7.1. Status of Components

The **Meta Database** is implemented in Oracle/8 [28]. The **Coordinator** consists of PL/SQL modules that execute the individual steps of the processes based on their execution graph. Our **Feature Extraction Engines** are able to extract texture features [27] and two different color features (histograms [16], and statistical values [42]). Furthermore, the extraction algorithms enhance the core features with spatial information by dividing the image into several (overlapping) regions [13, 14]. Currently, these algorithms employ 3 different partitioning schemes with 1, 5, and 35 regions. Altogether, our feature extraction engines can extract 9 kinds of feature vectors with a dimensionality varying from 9 to 2240. We have implemented two different **Data Repositories**: One that delivers the images as they are, and one that is network-aware [10]. Due to the high dimensionality of the features mentioned above, our **Index Service** uses VA-Files [47] to support efficient nearest neighbor similarity search. We have implemented three different **Client** interfaces. One is an HTML-based interface that allows to browse and search for similar images, but does not use the network-aware capabilities of our server. The second client is written in Java and allows users to comfortably enter their query and to exploit the network-awareness of our servers. The third client is based on Tcl/Tk. Each of the components is observed by a dedicated **Agent**. For example, the data repository agent monitors the contents of the repository. In case of an update (insertion or removal of images), it initiates the corresponding processes automatically. The agent database is also realized using Oracle8.

7.2. Status of the Image database

Our current database consists of 120,000 images collected from the web. To give an impression of the size of our image retrieval system, the following list enumerates the amount of data stored in Chariot:

- The 120,000 images consume around 10 GB of disk space.
- For each image, we have extracted 9 kinds of feature vectors as mentioned above. The total amount of feature data per image is 16,892 bytes. Hence, the feature data occupy 1.8 GB for all images.
- The index service maintains a VA-File for each feature. The size of vector data in the VA-Files varies between 5 and 1000 MB, and the size of approximation data varies between 1 and 250 MB, depending on the features used for the indexing. Searching for similar images with the VA-Files takes from 0.5 to 30 seconds, depending on the chosen index.

Insertion of N images with $k = 9$ features	1 FEE		7 FEEs	
	optimal	our DBHO	optimal	our DBHO
$N = 1$, 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 4. A summary of the measurements with our prototype system.

- The size of the meta-information on the image data (e.g. size, resolution, placement information, annotations) is 0.9 GB.

7.3. Benefits of Parallel Feature Extraction

In a number of experiments, we have dynamically added an extraction service while uploading a larger portion of our image database. After the initialization and registration phase, 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 distributed tasks evenly among the components. On the other hand, after the removal of a service, the coordinator re-distributed its tasks, and no longer sent new 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.

Table 4 summarizes the measurements of a bulkload and a single image insertion with our image retrieval system consisting of 7 feature services. The bulkload consisted of $N = 100,000$ images, for which 9 features had to be extracted. The total cost for the preprocessing of the bulkload was: $C(N, k) = 113.4\ days$. The table lists the optimal and measured times for the insertion with 1 and 7 feature extraction services.

7.4. Benefits of Parallel NN-Search

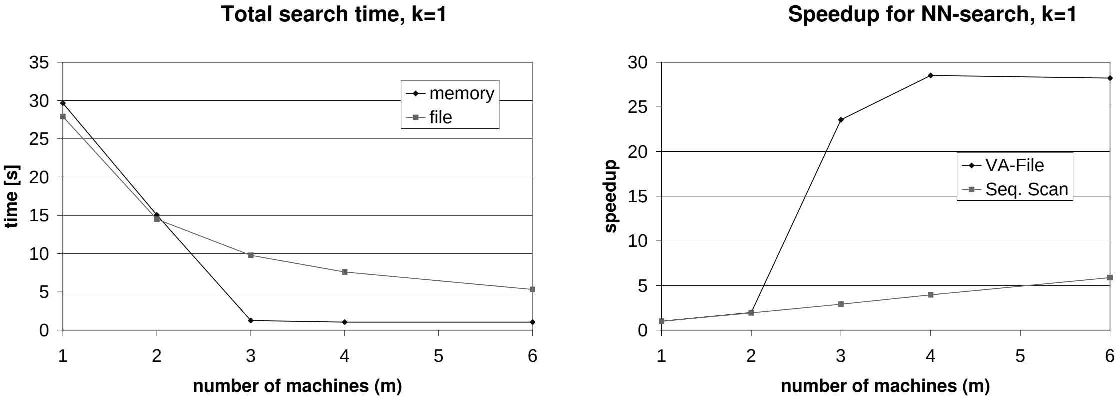
Figure 10 (a) compares the search time of parallel VA-File when searching for nearest neighbor for the two following techniques: (1) exploiting the memory available (denoted as *memory* in the figure), and (2) fetching data always from disk (denoted as *file*). The difference in the search times of the two techniques is not large when employing one or two machines. Namely, the approximation data (230 MB) does not fit into main memory. However, the effect becomes visible if the cluster consists of more than three machines. In this case, the search time of the *memory*-technique is less by a factor up to five. This is because disk IO is not necessary any more, i.e. the IO-bottleneck is eliminated.

Figure 10 (b) demonstrates that the VA-File profits more from this effect than the sequential scan. The figure compares the speedup of the response times with the VA-File and with the one with sequential scan. Since the vector data is 900 MB, a cluster with 6 machines does not suffice to hold the entire vector data in main memory. On the other hand, the approximation data, only 230 MB, fits into main memory with more than three machines. Therefore, the speedup of the VA-File increases significantly between $m = 2$ and $m = 4$. In Figure 10 (b), this effect is not visible for the sequential scan. We could observe it if the cluster consisted of more than 9 machines. Further, notice that the speedup of NN-search is almost 30 when using three machines instead of one. Instead of 30 seconds for NN-search on a single machine, parallel VA-File-based search with three machines lasts only about one second! Thus, we have achieved interactive-time similarity search with a very small number of machines already.

8. Conclusions

We have presented higher order databases as a vision for the future infrastructure of distributed information system engineering. Foundational work on composite transactions and transactional process management are the basis for the understanding of concurrent access in a complex composite system.

From a more practical side we have built a networked image retrieval system by coordinating specialized components. As it should have become clear in the second part of this paper, database technology helps in implementing such systems. Transactional processes and dynamic control of replication and availability has been achieved by implementing a DBHO with a database system. Nevertheless we see that a programmer using today's infrastructure has to deal with many low-level details. The situation today is



(a) (b)

Figure 10. Search time (a) and speedup (b) for parallel VA-File-based search and parallel sequential scan search as a function of the number of machines.

similar to the situation before modern databases were available.

Currently, all the processes of the coordinator of our image database are "hard-coded", i.e. there is a piece of code that describes step-by-step how a process is executed, and how the meta-database is accessed in order to guarantee transactional properties. Therefore as a next step we will factor out general steps from our special implementation and realize a DBHO that does automatic code generation for client process specifications. The vision of a DBHO goes even a step further than this: instead of programming processes, one describes them together with their transactional properties. When a new process is integrated into the coordinator, the according code is automatically generated and linked to the coordinator. We are convinced that generated code will execute many steps reliably and almost as efficiently as the system we have built manually.

Acknowledgments. Many discussions and joint work with Gustavo Alonso have influenced this direction. At early stages of the Cosmos project, contributions by Moira Norrie, Markus Tresch, Steven Blott, and Gerhard Weikum shaped the projects.

References

[1] G. Alonso, S. Blott, A. Fessler, and H.-J. Schek. Correctness and parallelism in composite systems. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, Tucson, Arizona, May 1997.

[2] G. Alonso, A. Fessler, G. Pardon, and H.-J. Schek. Correctness in general configurations of transactional components. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, Philadelphia, Pennsylvania, USA, May 31 - June 2 1999.

[3] G. Alonso, A. Feßler, G. Pardon, and H.-J. Schek. Transactions in Stack, Fork and Join Composite Systems. In *Proceedings of the International Conference on Database Theory (ICDT)*, Jerusalem, Israel, Jan. 1999.

[4] G. Alonso, C. Hagen, H.-J. Schek, and M. Tresch. Distributed Processing over Stand-alone Systems and Applications. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Athens, Greece, August 26-29 1997.

[5] S. Berchtold, C. Böhm, B. Braunmüller, D. Keim, and H.-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, USA, 1997.

[6] S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1996.

[7] P. A. Bernstein and E. Newcomer. *Principles of Transaction Processing*. 1997.

[8] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbour" meaningful? In *Proceedings of the International Conference on Database Theory (ICDT)*, 10–12 Jan. 1999.

[9] 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.

[10] J. Bolliger and T. Gross. A Framework-based Approach to the Development of Network-Aware Applications. *IEEE Transactions on Software Engineering*, 25(5), May 1998.

[11] M. L. Brodie. Que sera, sera: The coincidental confluence of economics, business, and collaborative computing. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Sydney, Australia, March 1999.

[12] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Greece, 1997.

[13] A. Dimai. Spatial encoding using differences of global features. Communications Technology, ETH Zuerich, Switzerland, 1996.

[14] A. Dimai. Differences of global features for region indexing. Technical Report 177, ETH Zürich, Feb. 1997.

[15] K. R. Dittrich and D. Jonscher. All together now—towards integrating the world's information systems. In *Advances in Databases and Multimedia for the New Century—A Swiss-Japanese Perspective*, 1999.

[16] 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), Sept. 1995.

[17] T. Grabs, K. Böhm, and H.-J. Schek. A document engine on a db cluster. In *High Performance Transaction Processing Systems Workshop 1999*, Asilomar, California, USA, September 1999.

[18] C. Hasse and G. Weikum. A performance evaluation of multi-level transaction management. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Barcelona, September 1991.

[19] IBM Corporation. http://www.ibm.com/.

[20] Informix Corporation. http://www.informix.com/.

[21] H. V. Jagadish. A retrieval technique for similar shapes. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, Denver, Colorado, 29–31 May 1991.

[22] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Tucson, Arizon USA, 1997.

[23] K. Kaufmann and H.-J. Schek. Extending tp–monitors for intra-transaction parallelism. In *Proc. of the 4th Int. Conf. on Parallel and Distributed Information Systems*, December 1996.

[24] H. Kitagawa, A. Morishima, and H. Mizuguchi. Integration of heterogenous information sources in infoweaver. In *Advances in Databases and Multimedia for the New Century—A Swiss-Japanese Perspective*, 1999.

[25] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. Technical Report CS-TR-3613, Institute for Advanced Computer Studies Dept. of Computer Science, Univ. of Maryland, College Park, MD, Mar. 1996.

[26] A. Makinouchi and S. Kuroki. Enhancing databases by introducing spatial data types for non-geographical applications. In *Advances in Databases and Multimedia for the New Century—A Swiss-Japanese Perspective*, 1999.

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

[28] Oracle Corporation. http://www.oracle.com/.

[29] L. Relly, H. Schuldt, and H.-J. Schek. Exporting database functionality - the concert way. *IEEE Data Engineering Bulletin, special issue on Interoperability*, 1998.

[30] U. Röhm, K. Böhm, and H.-J. Schek. Olap query routing and physical design in a database cluster. In *Proceedings of the International Conference on Extending Database Technology*, Konstanz, Germany, March 2000.

[31] M. Rys, M. C. Norrie, and H.-J. Schek. Intra-transaction parallelism in the mapping of an object model to a relational multi-processor system. In *Proc. of the 22nd Internat. Conference VLDB*, Mumbai (Bombay), India, September 1996.

[32] SAP AG, Walldorf, Germany. http://www.sap.com/.

[33] W. Schaad, H.-J. Schek, and G. Weikum. Implementation and performance of multi-level transaction management in multidatabase environment. In *Proc. of the 5th Int. Workshop on Research Issues on Data Engineering: Distributed Object Management, RIDE-DOM'95*, Taipei, Taiwan, March 1995.

[34] H.-J. Schek. Improving the role of future database systems. *Computing Surveys*, 28(4es), 1996.

[35] H.-J. Schek, G. Weikum, and W. Schaad. A multi-level transaction approach to federated dbms transaction management. In *Proc. of the First Int. Workshop on Inter- operability in Multidatabase Systems, (IMS'91)*, Kyoto, April 1991.

[36] H.-J. Schek, G. Weikum, and H. Ye. Towards a unified theory of concurrency control and recovery. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, Washington, DC, May 1993.

[37] 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.

[38] 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.

[39] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities - the lagunita report of the NSF invitational workshop on the future of database system research held in palo alto, california. *SIGMOD Record*, 19(4), February 1990.

[40] M. Stonebraker. Inclusion of new types in relational database systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Los Angeles, CA, Feb. 1986.

[41] 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.

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

[43] M. Tamura and M. Kitsuregawa. Dynamic load balancing for parallel association rule mining on heterogenous pc cluster systems. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1999.

[44] 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.

[45] R. Weber, J. Bolliger, T. Gross, and H.-J. Schek. Architecture of a networked image search and retrieval system. In *Eighth International Conference on Information and Knowledge Management (CIKM99)*, Kansas City, Missouri, USA, November 2–6 1999.

[46] R. Weber and H.-J. Schek. A distributed image-database architecture for efficient insertion and retrieval. In *Fifth International Workshop on Multimedia Information Systems (MIS99)*, Indian Wells, Palm Springs Desert, California, October 21–23 1999.

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

[48] G. Weikum. Principles and realization strategies of multi-level transaction management. *ACM Transactions on Database System*, 16(1), March 1991.

[49] G. Weikum and H.-J. Schek. *Database Transaction Models for Advanced Applications*, chapter 13: Concepts and Applications of Multilevel Transactions and Open Nested Transactions. Morgan Kaufmann, A.K. Elmagarmid (Ed.), San Mateo, CA, 1992.

[50] K. Yokota and T. Kunishima. Workflow modeling from a database point of view. In *Advances in Databases and Multimedia for the New Century—A Swiss-Japanese Perspective*, 1999.