Architecture of a Networked Image Search and Retrieval System*

R. Weber[1], J. Bolliger[2], T. Gross[2], H.-J. Schek[1]

[1]Institute of Information Systems [2]Institute of Computer Systems
ETH Zentrum, CH 8092 Zürich, Switzerland
Email: {weber,bolliger,trg,schek}@inf.ethz.ch

Abstract

Large scale networked image retrieval systems face a number of problems that are not fully satisfied by current systems. On one hand, integrated solutions that store all image data centrally are often limited in terms of scalability and autonomy of data providers. On the other hand, WWW-based search engines proved to be fairly scalable, and data providers retain their autonomy. However, such engines often confront users with links to servers that are not available or to images that no longer exist, i.e., they are unable to keep their meta-database consistent with the repositories' contents. Furthermore, existing solutions often neglect the cost of image delivery. The considerable variations in the effective bandwidth in today's Internet lead to highly unpredictable response times, which are often intolerable from the user's point of view.

This paper presents the architecture of Chariot, a networked image search and retrieval system that tackles these concerns. With respect to scalability and autonomy, Chariot follows the approach of WWW-based search engines by maintaining only the meta-data in a central database. Various specialized components (feature extraction, indexes, images servers) are coordinated by a middleware component that employs *transactional process management* to enforce consistency between the meta-data and all components. Moreover, Chariot incorporates mechanisms to provide more predictable response times for the image delivery over the Internet by employing *network-aware* image servers. These servers trade off the quality of the images to be delivered with the bandwidth required to transmit the images.

*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 Introduction

The use of image data to satisfy information needs is becoming more and more pervasive. There are plenty of examples and contexts where image data is an invaluable aid for information dissemination. To name a few examples: the print media is one of the largest consumers of topical image data provided by press photographers and news agencies; modern medicine depends heavily on images, such as MRIs, X-rays, or CAT-scans produced by diagnostic equipment; satellites produce a wealth of digital image data that is used by various scientific applications; advertisement and entertainment industries are customers of a wide range of digital images.

Many of these examples of producer-consumer relationships share a common set of concerns as far as the dissemination of the image data is concerned. One of the major concerns for *consumers* of digital image data is to find the relevant images "reasonably fast", even if they belong to different providers' collections. A consequence is that users do not want to search each provider's image collection separately. The requirements of (predictably) fast access and a single point of contact call for a comprehensive and global infrastructure to maintain, exchange and search for image data. Such an infrastructure, besides allowing for fast image search and retrieval by a widely dispersed set of users, must also account for the concerns of image *producers* (or providers), who often want to sell their image data to potential customers and yet must be able to retain their autonomy.

To date, there have been basically two approaches to build such networked image search and retrieval systems, both of which cannot fully satisfy these requirements:

Web-based image search engines, such as WebSeek [31] are fairly scalable (and can therefore provide fast access, even under heavy load), since they maintain only meta-data on images and let the image data reside in external, autonomous repositories. *Scalability* to large number of users and images is a necessary prerequisite to predictably fast information retrieval. However, these systems are often unable to keep their meta-database *consistent* with the repositories' contents. As a result, the user is confronted with links to images that no longer exist. Such inconsistent results can severely impact the user's experience of the search process, because a lot of time may be wasted hunting inexistent links/images.

On the other hand, integrated solutions based on database technologies, such as employed by PhotoBook [13] or QBIC [14], store all the image data centrally and hence are able to cope with the consistency problem. However, such an approach does not allow image providers to retain their autonomy. Moreover, restricted network and storage capabilities often limit scalability. These limitations impact the search time experienced by the user, because the user may have to contact multiple search engines to get satisfactory results or may have to wait long for an answer because the retrieval of image data (by other users) places a considerable burden on the system resources (needed for the image search).

In addition to the time required to locate images of interest, a considerable amount of time may be consumed by the retrieval of the images from the image repositories over the network. The fact that paths in today's Internet experience widely varying and fluctuating bandwidths leads to highly unpredictable response times, which are incompatible with the requirement for "reasonably fast" image search and retrieval. Current solutions that try to statically define and optimize image quality (and size) for users accessing the repositories over low-bandwidth links result in unnecessarily poor quality for users with high-bandwidth access. Images delivered in poor quality may not be of much use to the user, if key features are obscured so that the user is unable to assess the relevance of an image. None of the two approaches above address the issue of predictably fast image delivery.

Based on these observations, we present the architecture of Chariot, an image search and retrieval system, which is designed to support a large number of networked users, and which allows to efficiently search in large amounts of image data. The architecture addresses the following key requirements for such systems:

Scalability and autonomy: Chariot's distributed, component-based approach that separates the meta-data and image data management provides the necessary scalability with regard to the number of users and the number of images, yet allows image providers to retain their autonomy.

Consistency: Chariot's middleware builds on the principles of *transactional process management* and is therefore able to ensure availability and consistency in a fault-tolerant and flexible way.

Predictable response time: *Network-aware* image servers that adapt the quality of the images in response to fluctuations in the available bandwidth allow for the timely delivery of images.

Moreover, content-based image retrieval is a rapidly changing research area. Therefore, any image search and retrieval system using content-based retrieval methods must be *extensible* with respect to new feature extraction algorithms and indexing methods. In the Chariot system, the requirement of extensibility is accounted for by the modular, component-based architecture of the search engine.

To this end, we present the *architecture* of a networked image search and retrieval system[1], which addresses these requirements by a modular and component-based approach that separates the meta-data and image data management. Consistency is ensured in a fault-tolerant and efficient way by means of *transactional process management*. The problem of providing predictable response time for image delivery is tackled by adaptive (*network-aware*) image servers.

The remainder of the paper is organized as follows: Section 2 reviews the key requirements for networked image search and retrieval systems and introduces Chariot's distributed, component-based architecture and outlines two typical usage scenarios for such systems that exemplify the problem of maintaining consistency, which is addressed in Section 3, and the problem of providing predictable response times for the image delivery (Section 4). Furthermore, we report on the status of our implementation (Section 5) and on experiences with the current prototype (Section 6).

2 Distributed, component-based architecture

This section introduces the architecture of the Chariot image retrieval system. We justify our design decision by considering the requirements of scalability, autonomy and extensibility and argue why central solutions do not fulfill these requirements.

2.1 Design Considerations

Scalability. Consider a monolithic image retrieval system that maintains all data in a central database system. The key problem of this approach is that it does not scale well: The network capabilities of the central database bound the number of concurrent consumers,

[1] A description of the retrieval methods used in Chariot, as well as a discussion of their efficiency and efficacy are outside the scope of this paper (see [30, 12]).

and the storage capabilities bound the amount of image data. Consequently, a first step is to *distribute* the image data among several servers, and to maintain only *meta-information* about the images in the central database. As a result, our system consist of a central meta-database and several (external) image repositories. As a consequence, we can distribute requests for image data among the available servers such that each server is equally loaded. If the network or storage capabilities do not suffice, we may simply add new servers[2].

Autonomy. Consider an image provider who wants to charge customers for accessing its high-quality images. With the monolithic approach, such a provider must export its entire archive to the image retrieval system. In doing so, the provider can no longer control the distribution of the contents, and it is difficult to charge customers (the image retrieval system would have to charge customers on behalf of the provider). Hence, such a provider would rather like to export only the meta-data of the images to the retrieval system to retain autonomy and to limit the distribution of images to customers who pay for the items. The separation of image-data and meta-data allows image providers to retain their autonomy. Moreover, the advantage of a centralized approach remains: customers need not search for items at many places, but they can use a central service to satisfy their information needs. Some cooperation 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.

Extensibility. Content-based image retrieval relies on the extraction of specific features from the images, such as color [28, 27, 11], texture [20], or shape [17, 18] 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. Currently, commercial database system do not contain index techniques that provide fast similarity search in feature spaces. Hence, index structures like X-tree [3], M-tree [9], or VA-File [29, 30] must remain outside the database (as external index). Furthermore, the architecture must allow to exploit new indexing, browsing and searching techniques to provide fast and high-quality content-

[2]We are aware that reliability and scalability of the meta-database is an issue not treated here. It can be solved by standard database technologies.

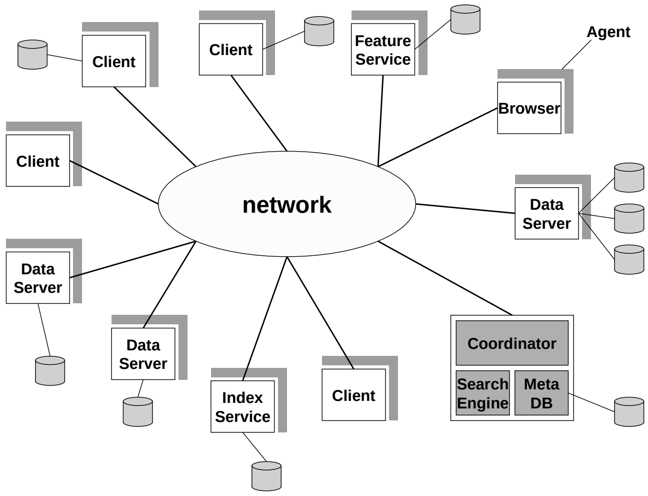

Figure 1: Components are autonomous; the middleware consists of the coordinator and the agents plugged into each of the components.

based retrieval. To ensure extensibility, feature extraction, index maintenance of feature values, searching and browsing should not be part of the central meta-database system. Rather, autonomous external components should provide these services. As with data repositories, coordination agents observe these components to provide status information such as availability and current load to the meta-database system. Such an organization allows, for example, to balance the load of feature extraction tasks among the available components and to replicate bottleneck components.

Coordination. In such a distributed, component-based architecture coordinating the components is no longer a simple task. Coordination is required to correctly react to changes in the contents of image repositories and to maintain consistency (this issue is addressed in Section 3). 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 direct clients towards image servers that allow for fast image delivery. Discussion of the latter two aspects is beyond the scope of this paper, however, these issues are accounted for in our system. The meta-database additionally stores information about availability of the components and bandwidth information for observed server-client paths.

Figure 1 depicts the component-based architecture of our image retrieval system that meets the scalability, autonomy and extensibility requirements. The central system consists of the meta-database, of a simple search engine, and of the coordinator. The coordinator's task is to maintain the meta-database and to keep it consistent with respect to the status and contents of all external components. The central system is one part of the *coordination middleware*. The other part are the 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 (data server), process meta-information (feature service, index service), or assist users to retrieve items of interest (browser, client).

2.2 Application scenarios

The component-based architecture presented in the previous section addresses the requirements of scalability, autonomy, and extensibility. In the following, we describe the efforts of our system with regard to the remaining requirements, namely availability, consistency and provision of predictable response times. To this end, we use two inherently different application processes: a) inserting a new image into a repository (InsertObject), and b) retrieving images that are similar to a given reference image (SimilaritySearch). The first process is used to illustrate how consistency is kept in the system (Section 3). While InsertObject is not time-critical, SimilaritySearch is time-critical in that a user waits for the result of the query. This process illustrates the measures taken to ensure predictable response times (Section 4). In this section, we sketch the relevant steps of the two processes.

Process InsertObject**:** 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.

Regarding the process InsertObject, the coordination middleware must guarantee that the process terminates (i.e., all steps are executed) and that the new image is visible only after the entire process is executed (isolation). The latter requirement is necessary to ensure that a user, who happens to issue an image search while the system is processing the insert operation, is not confronted with inconsistent results.

Process SimilaritySearch**:** The user issues a similarity search with a reference image. Since the reference image may not be part of the retrieval system, features for that image may not exist. Hence, the feature values required for the index lookup must be extracted. Subsequently, the search engine accesses several external indexes 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 image. 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 and the transmission costs are often negligible. This is not the case for image search and retrieval systems where the feature extraction of the query image, as well as the transmission of the relevant images can consume a substantial (often unpredictable) amount of time (e.g., due to bandwidth constraints). Therefore, our main concerns are an efficient and scalable architecture for feature extraction and the provision of predictable response times by means of network-aware servers.

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

3 Transactional process management

In the following, we exemplarily consider the case of an InsertObject process to discuss the issue of consistency. Here, consistency means that the meta-data maintained by the coordination middleware and the related indexes exactly reflect the contents of all external repositories. We first describe why traditional distributed transactions are not adequate for our architecture. Then we show how we can avoid these problems by establishing higher-layer transactional process management on top of traditional transaction management of the components.

3.1 Distributed transactions are not adequate

An agent may use a distributed transaction to update the meta-database according to the InsertObject process. Consider Figure 2: The agent starts a distributed transaction Trx after becoming aware of the newly inserted image. The first step is to create a new object in

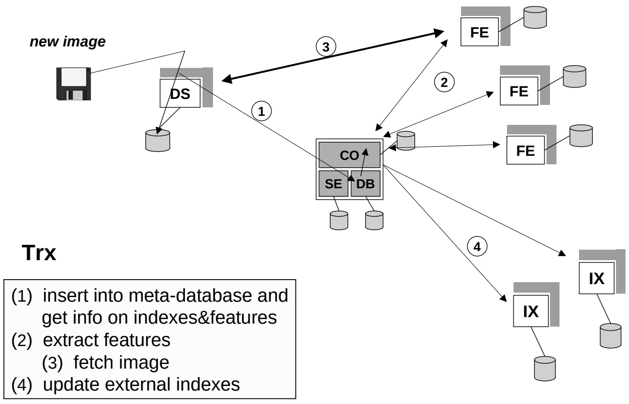

Figure 2: Executing InsertObject in a distributed transaction.

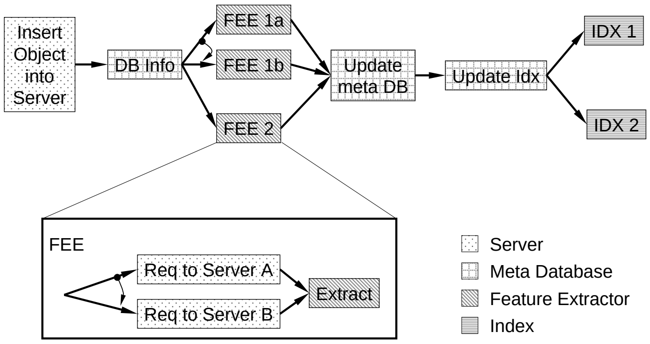

Figure 3: Execution graph for the InsertObject process

the meta-database (1). The meta-database then triggers the coordinator to extract the features of the new image with the help of several feature extraction engines (2). These engines request the image from the server (3), determine the feature values, and insert these values into the meta-database. If all features are determined, the coordinator requests all external index services to update their indexes (4). After the successful execution of steps 1-4, the agent commits the transaction, e.g., with a 2-phase-commit protocol.

Since we put all steps into a distributed transaction, we clearly have solved the consistency problem. However, this approach has serious drawbacks:

- All-or-nothing atomicity is not adequate: Either all steps are executed or none. In our case, any failure (e.g. unavailable index service or network problem) requires to rollback the entire work and to restart the transaction again. In doing so, we would lose all computed feature values and would have to re-compute them in the new transaction.
- Isolation is ensured at too low a level: Resources in all component systems (pages in the meta-database and in external indexes) are locked until the transaction commits. As a result, a client cannot search in parallel to an InsertObject process because it must wait until the locks on the resources are released, i.e., until the end of the (potentially long) insertion process. Furthermore, since feature extraction is time consuming, distributed transactions run for a long time and failures in the system are more likely to affect a transaction. Resource locks are held for a long time, and conflicts between two transactions become more likely. Due to the high probability of conflicts, the probability of (distributed) deadlocks increases, and transactions are more likely to fail due to breaking deadlocks.

Ensuring atomicity and isolation in the traditional transaction sense means that all components are entirely controlled by the coordination middleware. However, existing components like web- or ftp-servers do not allow such a strong external control. Such control also contradicts the autonomy requirement as discussed in Section 2. Hence, data providers must be allowed to update their repositories without being tightly coupled to a distributed transaction of the coordination middleware. The question is whether we can allow autonomous insertions into repositories yet avoid incomplete metadata and inconsistent indexes. The next subsection shows how an additional transactional coordination layer on top of traditional transactions, called *transactional process management* solves the problems.

3.2 Transactional process management

Instead of distributed 2-phase-commit transactions, we apply higher-order transaction management built on top of existing database transactions. As introduced in [25], we apply transactional processes for a description of coordination processes. 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 3 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

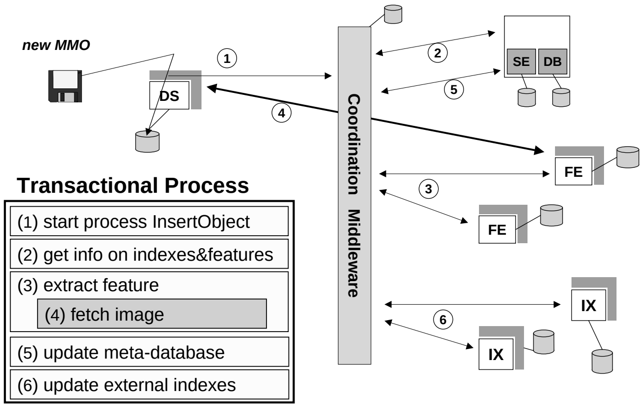

Figure 4: Executing InsertObject in a transactional process.

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 4 illustrates a transactional process for the InsertObject process. The obvious difference to Figure 2 (distributed transaction) is that the steps 1–6 are encapsulated by sub-transactions. 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 the Chariot 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 3. 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. Further details of the realization of the coordinator and of its meta-database using database technology (transactions, triggers, stored procedures) can be found in [4]. First evaluations are given in Section 6 below.

3.3 Related Work

In conventional database systems, recovery from failures and concurrency control are well understood problems. However, this is not the case when employing transactional processes with higher level semantics as is the case of the Chariot system. In [2], the authors have investigated concurrency control for workflow processes based on the assumption that all activities are successfully executed, i.e., they do not consider recovery in case of failures. HP's OPMS [8] is an open process management system that employs open and closed nested transactions for the control of workflow processes. In the context of Chariot processes, this approach is similar to ours, but it does not provide the required flexibility. For instance, in case of a failure, the recovery component should not rollback the entire work, but should find an alternative execution path. Coordination of components is also part of the more general project OPERA [1] at ETH Zurich. In OPERA, a complete integrated application-independent infrastructure for distributed application specification and execution is developed.

4 Network-aware image delivery

As we have seen in Section 2.2, the response time for an image search request depends on many (often rapidly changing) factors, such as server load and bandwidth availability. In today's Internet, the user has no control over these factors. Thus, it is important to provide as *predictable* a service as possible by allowing users to specify an upper bound on the time they are willing to wait for an answer. The system should tradeoff the quality of the results for the response time specified.

There are basically two steps in the process that allow the system to adapt to meet a limit on the response time: feature extraction on the query image and image delivery. In the first step, there is a limited amount of adaptation potential. In case of high load (or a short time limit), the system can select features that are cheap to extract (see Section 6). This choice may come at the cost of less accurate "comparison" and hence less relevant search results.

As the network is often the bottleneck resource, and since bandwidth reservation is not an option in today's Internet, image delivery must be adaptive (or *network-aware*) to meet the time limit. The following two problems must be addressed: (i) heterogeneity in network access (paths in the Internet span more than 6 orders of magnitude in bandwidth), and (ii) fluctuations in the available bandwidth (caused by congestion or intermittent connectivity). Without network-aware adaptation of the images to be delivered, image sizes must be statically defined and may, for example, be optimized for users accessing the image server across a (slow) wide-area network. This setup, however, leads to unsatisfactorily low quality for users with high-bandwidth access. This section outlines the approach taken in Chariot's *network-aware* image servers to ensure predictable response times. To understand the basic mechanisms of bandwidth adaptation, it is necessary to introduce the service model that captures the adaptation trade-offs to be made.

4.1 Service model

Image servers in Chariot act upon request messages containing a list of images to be retrieved and some restrictions on the service quality that characterize the minimum *quality* tolerable for the images delivered, the maximum quality that is beneficial for the user, and a limit T on the time allowed for processing the request and transmitting the response. The bounds on the quality may be (implicitly) imposed by the requester's processing or display capabilities. The quality of an image is defined by the resolution, color depth, the image format (e.g., JPEG, GIF), a format-specific parameter, such as JPEG's compression factor [16], and a (user-defined) weighting of these image characteristics.

The server's task is to deliver all the requested images to the client within time T, attempting to maximize the overall quality of the images transmitted while respecting the restrictions on the image quality. The range for dynamic adaptation to bandwidth availability is bounded by the minimal and maximal quality specified by the client. To quantify the server's task, a *quality metric* (i.e., a *utility function*) is defined as the weighted sum of the individual image qualities to be delivered. Weights for the quality metric characterize the relative importance or the relevance of an image. Note that information on relative importance of the images can, but need not, be provided by the search engine. Such information can take the form of relevance scores (between 0 and 1) or distance metrics, etc.. The network-aware server need not only dynamically adapt due to network service degradation (e.g., a drop in bandwidth) but should also try to opportunistically exploit extra bandwidth to deliver as many high quality images as possible within time T.

4.2 Adaptation to bandwidth changes

To adhere to the service model, network-aware servers must address the following questions: (i) how to find out about the currently available bandwidth on the path from the server to the client, and (ii) how to adapt the delivery process (i.e., the quality and hence the size of the images to be delivered) such that the objectives of the service model are met. (i) implies that the server must try to find out about the network path properties at the start of the transmission (to mitigate the problems imposed by heterogeneity) as well as track dynamic changes in network service quality during transmission of the images (to deal with bandwidth fluctuations). There are numerous strategies how to obtain feedback about network resource availability [5] (e.g., application-level monitoring [26], "promoting" information from the transport protocol [6] or the network layers [19]). A discussion of the pros and cons of each strategy is beyond the scope of this paper, and we simply assume for the remainder of our discussion that a mechanism is in place which allows the network-aware server to periodically obtain estimations of the bandwidth available.

We briefly sketch the mechanism of adaptation used in Chariot (see [5] for details): Given an estimate of the available bandwidth and information on the properties of the images to transfer (size, resolution, ...) the network-aware server must *periodically* compute the time needed for the delivery of the remaining images. If the time needed exceeds the time available, the server must choose a set of images and a set of image transformation algorithms that can reduce the size of the images (e.g., compression, scaling, ...) and recompute the time needed for the transfer of the images that are reduced in quality (and hence in size). Similarly, if all of a sudden there is ample bandwidth, reduction decisions from previous adaptation steps can be "undone" to improve the quality of the images delivered.

The adaptation algorithm employed by Chariot builds on two components: (i) a function f to estimate the time needed for the transfer of the remaining images, and (ii) a search for the optimal combination of images and reductions of these images that allows to meet the time constraints.

Function f takes the estimated bandwidth, the size of the transformed/reduced images, the (processing) costs for the reduction algorithms, as well as the server load into account to produce an estimate of the time needed. In addition to predictions on the server load and available bandwidth, the adaptation relies on models for the costs of and the size reductions achieved by an image transformation algorithm for a given image (such models are often quite easily obtainable [5]).

If adaptation is required to answer the request within the time allotted, the adaptation algorithm must find a subset of images to reduce and decide on applicable transformations that reduce the time needed for the transfer. Among all the combinations of image subsets and applicable reduction algorithms, the server must find those combinations for which the time needed (as computed by f) is within a given envelope of the time available (e.g., 5% of the time limit specified by the user). For all the combinations fulfilling this criterion the *utility function* is applied to compute the *quality* of the response. Finally, the combination with the highest quality (i.e., with the highest *utility* to the user) is chosen. The images that can be transmitted till the next adaptation point are then transformed (according to the adaptation decision made) and delivered to the client.

4.3 Related work

Recent work has shown that adaptation is an effective way of enhancing the user's perception of service quality in environments where resource reservation is not possible. Various systems have been built that dynamically adapt their behavior (their sending rate) in response to changes in the network: Odysee [23] pursues a client-based approach to adaptation and focuses on orchestrating multiple concurrent resource-aware applications on a (mobile) client. Fox et al. [15] use a proxy-based architecture employing so-called *distillation services* to enhance scalability of dynamic adaptation. Other systems, such as VIC [21] or IVS [7], have shown that continuous media streams in multicast sessions also benefit from adaptation. Their feedback-driven adaptation scales back quality and hence resource consumption when application performance is poor, and they attempt to discover additional resources by optimistically scaling up usage from time to time.

5 Implementation status

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

5.1 Components

The **Meta Database** is implemented in Oracle/8 [10]. 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 [20] and two different color features (histograms [14], and statistical values [28]). Furthermore, the extraction algorithms enhance the core features with spatial information by dividing the image into several (overlapping) regions [27, 11]. 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 as discussed in Section 4. Due to the high dimensionality of the features mentioned above, our **Index Service** uses VA-Files [29, 30] to support efficient nearest neighbor similarity search. We have implemented three different **Client** interfaces. One is a rough 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 server. 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.

5.2 Image database

Our current database consists of 105,000 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 105,000 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 1.65 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 4 and 900 MB, and the size of approximation data varies between 1 and 112 MB, depending on the features used for the indexing. Searching for similar images with the VA-Files takes from 0.2 to 20 seconds, depending on the chosen index.
- The size of the meta-information on the image data (e.g. size, resolution, placement information, annotations) is 0.85 GB.

Note: For a similarity search not the entire feature data has to be considered. Depending on the similarity

	1 FEE	9 FEEs
cheapest feature	5s	5s
most expensive feature	21s	21s
update of meta-data	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) using 1 and 9 feature extraction engines (FEE) in parallel.

model, only a fraction of this data is used. For example, if a user is interested in color-similarity, only 4 MB of feature data are searched. However, if the user requires color- and texture-similarity using a partitioning scheme with 35 regions, 1.2 GB of feature data must be considered to answer the query. The differences in the amount of feature data to be searched allow the search engine to trade off the search time to the quality of the search results.

6 Experiences

Given the Chariot prototype implementation, we have observed that the transactional process management exploits parallel resources almost optimally, and that network-aware servers efficiently trade off the search time to the quality of images. In the following, we discuss these two findings in more details.

6.1 Transactional process management

As discussed in Section 3, transactional process management has many advantages over traditional approaches: the ability to handle failures, as well as its flexibility with respect to isolation and atomicity. In the following we will evaluate how we can exploit transactional process management for increased efficiency by parallel executions on replicated components. We use a cluster of workstations, consisting of PCs (300 Mhz Pentium II, 128 MB), and compare parallel insertions with sequentially inserting one image after the other (called the naive approach). For that purpose, we have first performed a preliminary experiment that provides us with figures for the cost of inserting a single new image. We measured the time to insert an average-sized image (resolution 1024x796) with two different settings: 1) only one feature extraction engine (1 FEE), and 2) 9 feature extraction engines (9 FEEs). The results are presented in Table 1. Notice, that the cost of extracting a single kind of feature vector varies between 5 to 21 seconds. Compared with the 4 seconds necessary to update the meta-database, we see that the time to insert an image is dominated by the feature extraction effort. Therefore with 1 FEE the time to insert one image is the sum of the times spent in every component, in our configuration 102 seconds. If we have as many FEEs as we have kinds of feature vectors, in our configuration 9 FEEs, the time for one image insertion is dominated by the time of the most expensive feature extraction plus the time for the update of the meta data, i.e. 25 seconds (see Table 1). While this improvement from 102 to 25 seconds, possible by allowing intra-process parallelism, is significant for the insertion of a single image, it is still naive for the case of many image insertions in a bulkload step: For the insertion of N images we have N times 25 seconds. However the sequential execution of N insertions is unnecessarily restrictive and has the following drawbacks: If we have n FEEs and k kinds of feature vectors, and if $n \geq k$, not all engines are used by the InsertObject process. Even when we have less FEEs than kinds of features, the FEEs are not equally utilized due to the large differences in the feature extraction times (between 5 and 21 seconds in our experiment). Moreover, during the update of the meta-database, none of the FEEs is occupied.

Employing transactional process management ensures correct interleavings at the coordinator level and — in case of insertions — allows to almost unrestrictedly execute all steps of different InsertObject processes in parallel, i.e., inter-process parallelism is performed in addition to intra-process parallelism used in the naive approach. Due to the coordinator's (centralized) control over the processes, it maintains a table that reflects the current load of the FEEs. The required extraction requests are evenly distributed among the available FEEs (i.e., load balancing of feature extraction requests). Moreover, while the features of some images are being extracted by the FEEs, the coordinator concurrently updates the meta-database with the information of images whose features are already computed. Consequently, the cost for inserting N images is only given by the cost of extracting the features for the images. Employing this load balancing as implemented in the Chariot prototype, the time for the insertion of a stream of N images can be estimated by $(N \cdot 98s)/n$. All n FEEs are occupied over the entire period of the insertion task. An interesting experiment in our Chariot system was the upload of the entire repository, i.e., the insertion of 105,000 images. We employed 7 FEEs (i.e., $n = 7$) and one image server. We extracted 9 kinds of feature vectors, and each FEE was able to extract any of the 9 kinds. The measured times for the complete upload were as follows: Each of the 7 FEEs was extracting features during 281 hours each, and all FEEs were equally loaded. The cost of building the meta-database and the external indexes was of the order of several hours, and the cost of the coordination, i.e., execution of processes, network overhead, image delivery to engines, and load balancing of extraction task, was around 120 hours. Since, as expected, feature extraction dominates

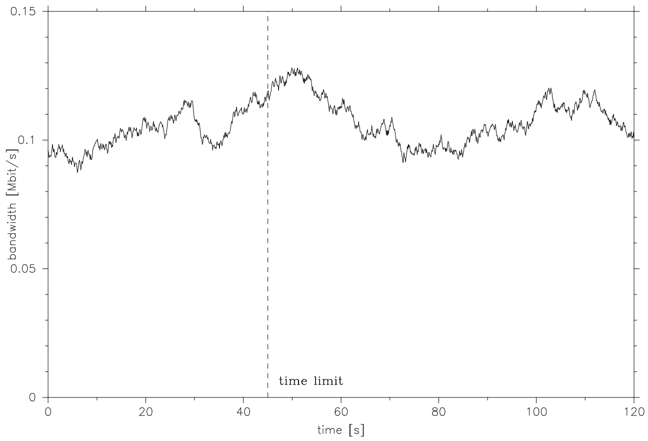

Figure 5: Bandwidth available for image transfer.

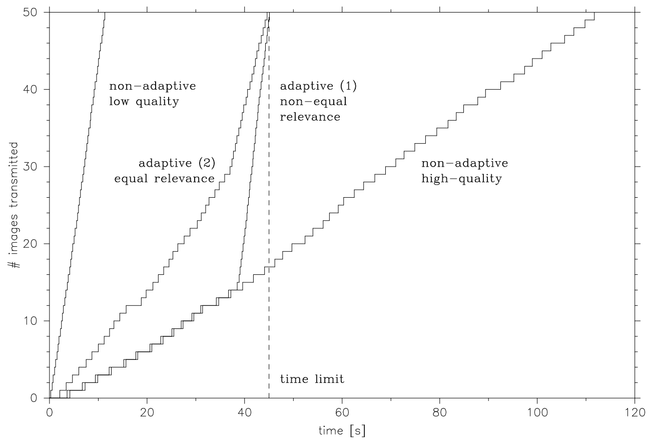

Figure 6: Comparison of adaptive and non-adaptive image delivery: Progress of transfers.

the waiting time and since the components worked in parallel, the total waiting time for completely inserting 105,000 images was 12 days. This compares favorably with the naive approach where we would have needed more than 70 days.

6.2 Network-awareness

To show that network-aware image servers are able to fulfill the service model defined and to illustrate problems that need to be addressed in the future, we conduct a simple experiment within Chariot. A user issues a search for images that are similar to a given query image[3]. Within 45 seconds[4] the user wants to see a list of 50 JPEG images with a minimal resolution of 180×120 pixels and a minimal JPEG quality factor of 25. The images are stored at the server in the JPEG format (default quality factor 75) and have a resolution of 720×480 pixels.

We compare three alternatives for image delivery: (i) non-adaptive, high-quality image delivery, optimized for users with high-bandwidth access to the image server (720×480, quality 75), (ii) non-adaptive, low-quality image delivery optimized for low-bandwidth access (180×120, 25), (iii) adaptive image delivery as described in Section 4. For the adaptive image delivery, we evaluate two alternatives that differ in their definition of the *utility function*, more precisely, in their interpretation of the relevance scores provided by the search engine. Recall from Section 4.1 that the utility function is defined as weighted sum of the individual image qualities, where the weights are defined by the "relevance" of an image. For both alternatives, we use the image size as a metric for the quality of an image, that is, we simply assume that the quality of image i is proportional to the image size s_i (in bytes)[5]. The first alternative defines the utility of image i as $u_i = \lfloor 10 \cdot (r_i - r_{limit}) \rfloor \cdot s_i$, where r_i is the relevance score assigned by the search engine ($r_i \in [0, 1]$) and r_{limit} denotes a limit on these scores below which a image is considered irrelevant (zero utility to the user). (In our experiment there were no such images). The second alternative makes the assumption that all the images requested are equally relevant to the user, that is, $u_i = s_i$ (this may be the case if a user simply browses the image repository with no particular goal in mind). The choices made here are fairly arbitrary and are just used to illustrate the influence of the relevance information on the behavior of the adaptive image delivery.

To ensure reproducibility of the experiments we use trace modulation (see [24]) to emulate the characteristics of a two-minute transfer over a ISDN dial-up line on a LAN-environment. The average bandwidth is 106kbps (see Figure 5).

Figure 6 shows the progress of the image transfers as a function of time. The y-axis denotes the number of images transferred. The figure shows how non-adaptive, high-quality image delivery takes considerably longer than the 45 seconds the user is willing to wait for them in our example. On the other hand, non-adaptive, low-quality image delivery, optimized for server-access over (non ISDN) modem-lines terminates considerably faster and thus under-utilizes the resources available. Network-aware delivery is able to meet the time limit fairly well (in both scenarios considered).

Figure 6 further illustrates that the behavior of

[3]We use color features, enhanced with spatial information, for the similarity search (see Section 5 and [27]). The query image and results are the same as in [27] (Figure 2).

[4]The time limit for the entire search process is set to 55 seconds. The 10 seconds used for feature extraction on the query image, the indexing and communication between client and search engine have been subtracted before issuing the request to the image server.

[5]This assumption is made only to simplify the presentation. Note that this may not be the most appropriate choice, as indicated by a user study on the perception of image quality [22].

	time		data		utility
	[s]	[%]	[kB]	[%]	[%]
non-adaptive (high)	111.7	100.0	1491	100.0	100.0
non-adaptive (low)	11.4	10.2	114	7.6	7.6
adaptive (1)	45.1	40.4	565	37.9	51.4
adaptive (2)	44.6	39.9	559	37.5	—

Table 2: Summary of adaptive and non-adaptive image delivery.

adaptive image delivery depends strongly on the utility function used. The first alternative (curve "adaptive (1)") delivers high-quality images up to $t = 38s$ (the progress curve is in parallel to the "non-adaptive, high quality" curve). After that point images are sent in lower quality. This behavior clearly reflects the choice of the utility function. Since images are delivered in decreasing order of relevance, and since the relevance is the main factor in the utility function to be maximized, the server tries to pass on reductions to images that are less important. For the second alternative, where all the images are considered equally important, we see that the progress curve ("adaptive (2)") increases more steadily, which means that the reductions are spread out more evenly across all the images.

The results are summarized in Table 2. Compared to the non-adaptive, high-quality scenario, adaptive image delivery is able to meet the time limit. Both alternatives of adaptive delivery transmit only about 38% of the data. Relating this to the utility of the delivered images, we see that, using the utility function of the first alternative, adaptive transmission achieves a comparably high utility of more than 50% (column "utility"). The utility values for the second alternative are depicted by the "data" columns.

These results indicate that network-aware image delivery can indeed provide (more) predictable response times by trading off the quality of the images delivered. However, the behavior of adaptive image delivery depends strongly on the utility function used. In this respect, our paper merely presents a framework for experimentation but takes no position on what utility functions should be used. More work is needed to define appropriate utility functions, i.e., utility functions that are meaningful to users in other application scenarios. To pursue this goal, we need to (better) understand user concerns: What is relevant to a user when searching for similar images? How do reductions in image quality (achieved by the various compression schemes) relate to reductions in utility? As far as the second issue is concerned, the user study performed by McIlhagga et al. [22] may serve as a start.

7 Concluding remarks

The system described here uses database technology and networked components to provide an integrated architecture for an image search and retrieval system that addresses a wide range of issues such as autonomy, availability, scalability, inter-component consistency, and network-awareness. The architecture we have presented is a result of several projects at ETH Zurich. Evaluations and experiences of our implementations confirm our design decisions. In particular we have put emphasis on the coordination of the networked components by transactional process management that allows not only consistency between components but also allows to replicate bottleneck components such as image feature extraction services and index engines. We have integrated a very efficient and parallelizable way to support similarity search based on feature approximation and incorporated network-aware servers that trade off the quality of images to be delivered with the bandwidth required for transmission.

The architecture presented in this paper provides a first step towards a system that can be used on a global-scale in the Internet. We see many opportunities for improvement, but already in its current incarnation, the system serves our need for a working platform. Only an operational system can be used to explore the dynamic properties that are encountered by a distributed application, and given the large number of issues that must be addressed, experimental validation through deployment of an actual system is necessary. The interaction of database technology and networks, and the integration of network information into the realm of the information system, remain active research topics.

Acknowledgments. S. Blott and M. Stricker contributed to the design of the architecture and we appreciate their contributions. D. Ma, D. Paschoud as well as many students in their Diploma thesis work participated in the implementation effort. We further thank A. Dimai for providing us with feature extraction algorithms, and H. Schuldt for many helpful discussions on the issues of the paper.

References

[1] G. Alonso, C. Hagen, H.-J. Schek, and M. Tresch. Distributed Processing over Stand-alone Systems and Applications. In *Proceedings of the 23rd VLDB conference*, Athens, Greece, August 26-29 1997.

[2] I. Arpinar, S. Arpinar, U. Halici, and A. Dogac. Correctness of Workflows in the Presence of Concurrency. In *Proceedings of the Next Generation Information Technologies and Systems Conference (NGITS'97)*, Israel, June 1997.

[3] S. Berchtold, D.A. 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)*, pages 28–39, 1996.

[4] 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.

[5] J. Bolliger and T. Gross. A Framework-based Approach to the Development of Network-Aware Applications. *IEEE Transactions on Software Engineering*, 25(5):376–390, May 1998.

[6] J. Bolliger, T. Gross, and U. Hengartner. Bandwidth modelling for network-aware applications. In *Proceedings of IEEE Infocom '99*, page (to appear), March 1999.

[7] J-C. Bolot, T. Turletti, and I. Wakeman. Scalable feedback control for multicast video distribution in the Internet. In *Proceedings of ACM SIGCOMM '94*, pages 58–67, London, August 1994.

[8] Q. Chen and U. Dayal. A Transactional Nested Process Management System. In *Proceedings of the 12th International Conference on Data Engineering (ICDE '96)*, pages 566–573, 1996.

[9] 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)*, Athens, Greece, 1997.

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

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

[12] A. Dimai. *Invariant Scene Representation for Preattentive Similarity Assessment*. PhD thesis, ETH Zürich, May 1999.

[13] A. Pentland et al. Photobook: tools for content-based manipulation of image databases. In *Storage and Retrieval for Image and Video Databases, SPIE*, pages 34–47, 1994.

[14] 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, September 1995.

[15] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. In *ASPLOS-VII Proceedings, 7th Intl Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–173, October 1996.

[16] Independent JPEG Group. *IJG's JPEG Software Release 6a*. Independent JPEG Group, February 1996.

[17] H. V. Jagadish. A retrieval technique for similar shapes. In James Clifford and Roger 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.

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

[19] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. In *7th IEEE Symposium on High-Performance Distributed Computing*, pages 189–196, July 1998.

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

[21] S. McCanne and V. Jacobson. vic: A flexible framework for packet video. In *Proceedings of ACM Multimedia '95*, pages 511–522, San Franscisco, CA, November 1995.

[22] M. McIlhagga, A. Light, and I. Wakeman. Towards a design methodology for adaptive applications. In *Proceedings of ACM Mobicom '98*, pages 133–144, October 1998.

[23] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium of Operating System Principles*, October 1997.

[24] B.D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz. Trace-based mobile network emulation. In *Proceedings of ACM SIGCOMM '97*, pages 51–62, September 1997.

[25] 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.

[26] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson. RFC 1889: RTP: A Transport Protocol for Real-Time Applications. Request for Comments, Internet, January 1996.

[27] 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.

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

[29] R. Weber and S. Blott. An approximation based data structure for similarity search. Technical Report 24, ESPRIT project HERMES (no. 9141), October 1997.

[30] 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.

[31] WebSeek. http://www.ctr.columbia.edu/~jrsmith/.