Distributed and Parallel Databases, 9, 67–92, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Distributed Processing of Similarity Queries

APOSTOLOS N. PAPADOPOULOS apostol@delab.csd.auth.gr
YANNIS MANOLOPOULOS yannis@delab.csd.auth.gr
Data Engineering Research Lab., Department of Informatics, Aristotle University, 54006 Thessaloniki, Greece

Received July 23, 1997; Revised November 6, 1998; Accepted May 21, 1999

Abstract. Many modern applications in diverse fields demand the efficient manipulation of very large multidimensional datasets. It is evident, that efficient and effective query processing techniques need to be developed, in order to provide acceptable response times in query processing. In this paper, we study the processing of similarity nearest neighbor queries in large distributed multidimensional databases, where objects are represented as vectors in a vector space, and are distributed in a multi-computer environment. The departure from the centralized case embodies a number of advantages and (unfortunately) a number of difficulties that need to be successfully overcome. In this perspective, four query evaluation strategies are presented, namely **Concurrent Processing (CP)**, **Selective Processing (SP)**, **Two-Phase Processing (2PP)** and **Probabilistic Processing (PRP)**. The proposed techniques are compared analytically and experimentally, in order to discover the advantages of each one, as well as the best cases where each one should be applied. Experimental results are presented, demonstrating the performance of each method under different parameters values. Also, we investigate the impact of derived data that should be maintained in order to process similarity queries efficiently.

Keywords: distributed databases, multidimensional data, similarity queries, query processing

1. Introduction

Multidimensional data appear in many modern applications in diverse fields. Geographical Information Systems (G.I.S) require the storage and manipulation of objects in 2-d and 3-d spaces [14, 19]; Image/Video Retrieval by Similarity and Pattern Recognition require the extraction of features from the images and the mapping of these features in a high dimensional space, in order to speed up query processing [11, 17]; in Time Series databases, the objects are first transformed (e.g. by means of the Fourier transform) and then some components (these with the highest energy) are used to index the underlying set [2, 10]; documents in a Text Database can be represented as vectors (e.g. using the Latent Semantic Indexing technique) in a high dimensional space [8]; records in traditional alphanumeric databases can be viewed as points in a high dimensional space, assuming one dimension for each record attribute [16].

These applications require databases that are huge in volume. Often, multiple computer systems are used in order to support efficient and effective retrieval. In some cases, due to the nature of the application, there is no alternative except distribution (e.g. in medical information systems, where data corresponding to one hospital are apart from the data regarding another hospital). The departure from the centralized case offers a number of significant advantages such as: parallelism exploitation during query processing, distributed

control, incremental growth, higher availability and reliability. On the other hand, data distribution embodies a number of difficulties such as: complicated query processing and optimization strategies, communication cost overhead, load balancing considerations in order to avoid bottlenecks. These difficulties need to be overcome in order to support efficient manipulation and retrieval [7, 27].

In the aforementioned applications many types of queries may be posed by users. The most common query is to determine if a specific object O_q exists in the database. A more difficult query, that requires a substantial amount of system resources, is the similarity query. An object O_q is given and the k objects O_j that are closer to O_q are requested. The objects O_j, $1 \leq j \leq k$, are called the **nearest neighbors** of O_q. Furthermore, several restrictions may be applied. For example, we may be interested in the k nearest neighbors of O_q such that no O_j is further/closer than a specified threshold distance. Either in the simple or the restricted case, similarity queries are extensively used and efficient processing strategies are required.

In this paper, we study the similarity query problem in a distributed system. Several factors should be taken into consideration, such as the type of derived data, CPU cost, I/O cost, communication cost, degree of parallelism, throughput and acceptable response times. In the next section we present the appropriate background in multidimensional spaces, the motivation behind this work and the several assumptions introduced in order to approach the problem. In Section 3 similarity query evaluation strategies for distributed systems are presented and compared analytically with respect to the expected response time of a query. In Section 4 the importance of derived data is stated and their impact in query processing is investigated. The query evaluation strategies are compared experimentally in Section 5. Section 6 contains some useful discussion on major topics and, finally, Section 7 concludes the paper.

2. Background

2.1. Spaces

Given a number of objects (images, text documents, video clips, sounds, geometric entities) we may define several spaces that these objects could be embedded:

- **Vector space.** In an n-dimensional vector space each object is represented by a well defined set of n values, that correspond to a vector. For example, a 256-color image can be represented as a vector in the 256-d space, using the color histogram of the image. The similarity measure between two objects O_i and O_j, $d(O_i, O_j)$ could be, for example, the Euclidean distance between the corresponding vectors.
- **Metric space.** There are applications (e.g. in DNA sequences) where each object can not be represented as an n-d vector, but we can define a metric $d(O_i, O_j)$ between two objects O_i and O_j such that:

  1. $d(O_i, O_j) \geq 0$ (positivity)
  2. $d(O_i, O_j) = d(O_j, O_i)$ (symmetry)
  3. $d(O_i, O_j) \leq d(O_i, O_k) + d(O_k, O_j)$ (triangular inequality)

  It is evident that a vector space is also a metric space, but the inverse is not always true.

- **Non-metric space.** In a non-metric space it is impossible not only to define an n-d vector representation of the objects, but also to define a metric. This is because of the intuitive meaning of similarity in some applications where is very difficult to express similarity by means of a function between two objects. Examples can be found in the pattern recognition literature [25].

Fortunately, many real datasets are based on the vector space model. Although there are applications where the objects have no straight-forward vector representation, the work of Faloutsos and Lin [11] presents a method that can be effectively applied to derive a vector representation of a metric space. Therefore, it is reasonable to focus on vector spaces (or multi-dimensional spaces) in order to study the similarity query problem.

2.2. Nearest neighbor queries

Definition 2.2.1. Given an object O_q, a query requesting the k objects O_j closer to O_q (w.r.t. a distance metric) than any other object in the database, is defined as a k-nearest-neighbor query (k-NN query for short) or a k-similarity query.

Notice the difference between a similarity query and a range query. In a range query, an object O_q is given and all objects that lie in the circle with center O_q and a specified radius R are requested. Therefore, in the range query the distance is known whereas the number of objects retrieved is unknown. On the other hand, in a similarity query the number of retrieved objects is known whereas the distance is unknown. Evidently, a similarity query can be answered by means of repetitive applications of a range query, but this can result in unnecessary resource consumption in case we do not provide a suitable distance a priori.

Several research efforts have been performed, in order to provide efficient and effective processing techniques for the similarity query problem: In [13] the problem of nearest neighbor searching is studied under the optimized k-d-tree data structure. In [5] optimal expected time algorithms are reported for solving a number of closest point problems in the n-d space. Theoretical bounds are established and the authors show how a practitioner can benefit from these techniques. In a more recent paper White and Jain [28] examine the similarity query problem and propose techniques for designing variations of R-trees and k-d-trees more suitable for high dimensions. Also, in [6] a data structure (the X-tree) is presented that is specifically designed for high dimensional data. Experimental results show that the structure outperforms R*-trees [4] and TV-trees [20] by factors. In [21] we present techniques to answer nearest neighbor queries in declustered R-trees. Arya et al. [3] present the impact of taking into consideration boundary effects in the analysis of nearest neighbor queries. In [22] we provide expected upper and lower bounds in the performance of nearest neighbor queries in R-trees, by taking into consideration the fractal dimension of the dataset. Also, in [23] algorithms for similarity search in disk arrays are presented. An interesting approach that applies when there are multiple systems (databases) with different similarity measures is proposed in [12]. Due to a large number of applications, there is an ongoing research in nearest neighbor queries in high dimensional spaces, where it is much more difficult to provide efficient processing methods than for low dimensionalities.

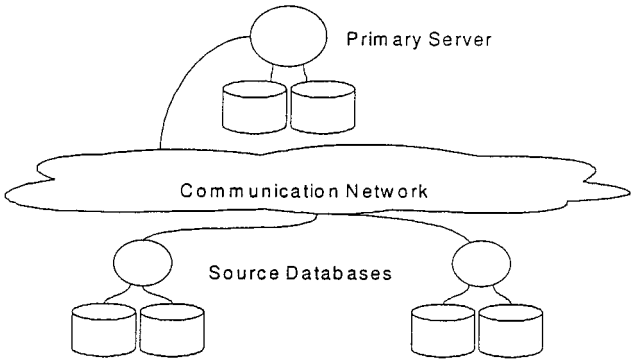

Figure 1. The abstract system architecture.

2.3. Motivation and assumptions

In this subsection we present the abstract architecture of a distributed multi-dimensional database, and discuss the basic motivation as well as the assumptions adopted. In figure 1 the abstract architecture is illustrated. The system is composed of a primary server that operates as a coordinator for the m source databases. All systems are communicating via a network configuration.

The primary server may be a data warehouse or simply a system that is responsible for controlling and supervising the source databases. We assume that query requests are initiated by a user's system and then submitted to the primary server for evaluation. Also, the query results are gathered from the source databases to the primary server and then are shipped back to the appropriate user's system. Despite the fact that we perform a distinction between primary and secondary sites, any secondary site could take responsibility of evaluating user queries. Each source database has complete control over the objects that it stores. Therefore, different access methods and optimization techniques may be utilized by the different databases.

Definition 2.3.1. Given a k-NN query Q, the response time for Q is defined as the time elapsed from query submission to query completion.

The challenge is to determine an efficient method for NN query processing in a distributed system. Moreover, the number of parameters is quite large and in some cases trade-offs occur (e.g. degree of parallelism vs. number of transmitted objects). The problem we are going to deal with in the remainder of the paper is stated as follows:

Problem Statement. Given a distributed multi-dimensional database and a *k-NN* query Q_k, find an efficient evaluation strategy, in order to minimize the response time of Q_k and to consume as few overall system resources as possible.

In order to approach the problem from a theoretical point of view, several simplifying assumptions should be introduced, resulting in a more feasible and tractable analysis. The basic assumptions introduced are summarized below:

1. Although we do not require the source databases to be homogeneous, we will assume that the cost to answer a given query is the same, for all source databases.
2. We assume that the similarity metric between two multi-dimensional vectors is the Euclidean distance (L_2 metric),[1] and every database respects this similarity measure.
3. The data are partitioned to the source databases in such a way that no replication exists. In other words, each object is stored in only one database.
4. If during processing we must retrieve P disk pages from a source database, the required time is $P \cdot T_p$, where T_p is the expected page access time [1].

3. Query evaluation strategies

3.1. Algorithms

Let a k-NN query, Q_k, be submitted for evaluation to the primary server. Our first approach is to examine the evaluation of the query when no derived data are available. In a following section, we discuss what kind of derived data are necessary to improve the efficiency in similarity query processing. We could define two extreme strategies to answer the query:

Concurrent processing—CP: Submit the query to all m source databases and collect k objects from each one. Among the $m \cdot k$ objects, select the best k (those that are closer to the query object).

Selective processing—SP: First activate one source database. Collect the best k answers. Send only the distances of these k objects to the next source database and collect another l objects, where $0 \leq l \leq k$. Continue until all source databases are visited and the best matches have been determined.

We note that the first method tries to maximize parallelism but retrieves too many objects ($m \cdot k$), whereas the second method, performs a more refined search, but no parallelism is exploited. Therefore, we define the next method, which is a combination of the two previous ones:

Two-phase processing—2PP: First visit f source databases and collect $f \cdot k$ objects. Then, select the best k and send the k distances to the rest $m - f$ source databases. Finally, collect the answers and determine the final set of nearest neighbors.

Finally, we define **Probabilistic Processing (PRP)** which performs an optimistic search, pretending that each source database will contribute with almost the same number of objects.

Probabilistic processing—PRP: First request $k/m+1$ objects from each source database. Then, formulate the current set of best matches, and if there are sources that are still relevant, visit them again and collect the final set of objects.

By requesting $k/m+1$ objects from each database, we can reject a database if the $(k/m+1)$-th distance from the query point is larger than the best k-th distance determined so far.

3.2. Theoretical study

We proceed with some theoretical investigation, regarding the efficiency of the four query evaluation strategies. The results will illustrate which strategy shows the best performance under what parameters, which are the advantages and disadvantages of each one.

Table 1 presents the basic symbols and the corresponding definitions that are extensively used throughout the paper. We denote with $R_j(k)$ the average query response time in seconds, for strategy j, in order to answer a k-NN query. The total processing cost comprises of three basic parts: CPU cost, I/O cost and communication cost. We expect that CPU cost will have a small impact on the performance comparison of the strategies and therefore, it is excluded from our theoretical study. However, CPU cost is included in our experimental study presented in Section 5.

Table 1. Symbols, definitions and corresponding values.

Symbol	Description	Value
m	number of source databases	5–30
N	total number of objects	100,000–10,000,000
N_j	number of objects in j-th source database	N/m
D	dimensionality of the vector space	2–20
S_n	size of a number in bytes	4
S_o	average size of an object in bytes	100–100,000
S_v	size of a D-dimensional vector in bytes	$D \cdot S_n$
S_p	size of a disk page in bytes	4K
S_{header}	size of a network packet header in bytes	24
S_{pmax}	size of a network packet in bytes (without header)	1500
T_p	page read time in seconds	0.01
k	number of nearest neighbors requested	1–500
C_j	contribution of j-th source database	
NC_j	net contribution of the j-th source database	
$R_j(k)$	query response time (in seconds) for strategy j	
NS	network speed in bytes per second	100,000–1,000,000
f	visited databases in step 1 of 2PP	1

Definition 3.2.1. We call the contribution C_j of the j-th source database, the number of objects processed and transmitted during the evaluation of a k-NN query. Obviously, $C_j \leq k$ for all j, where $1 \leq j \leq m$ and $\sum_{j=1}^{m} C_j \geq k$.

Definition 3.2.2. We call the net contribution NC_j of the j-th source database, the number of objects from the j-th database that participate in the answer set of a k-NN query. Obviously, $0 \leq NC_j \leq k$, for all j, where $1 \leq j \leq m$ and $\sum_{j=1}^{m} NC_j = k$.

Note that the contribution of a source database depends on the visiting sequence. Evidently, the net contribution of a source database is independent of the visiting sequence and depends on the data placement and the location of the query point. Under the uniformity and independence assumption, we expect that the net contribution of each database equals k/m.

Definition 3.2.3. The local processing cost[2] of a source database to process a k-NN query is defined as:

$$Cost_{\text{db}} = \left(\text{INA}(k) + \frac{S_{\text{o}}}{S_{\text{p}}} \cdot O \right) \cdot T_{\text{p}} \tag{1}$$

where INA(k) refer to the number of index node accesses for k nearest neighbors, which depends on the population of the database, the space dimensionality and the data structure used to store and manipulate the objects, T_{p} is the page read time, S_{o} is the average number of bytes per database object, S_{p} is the number of bytes per disk page and O is the number of objects that are accessed. We note that the first part of the above equation is due to the index search, whereas the second one is due to access of the objects' detailed description.

Definition 3.2.4. The cost for transmitting B bytes using the communications network is defined as follows:

$$Cost_{\text{trans}}(B) = \frac{1}{NS} \cdot \left(B + \frac{B}{S_{\text{pmax}}} \cdot S_{\text{header}} \right) \tag{2}$$

where NS is the speed of the network in bytes per second, S_{pmax} is the maximum capacity of a network packet, and S_{header} is the size of a packet header in bytes.

Based on the assumptions and the definitions given, let us proceed with a comparative study among the four methods described in the previous paragraphs. For each strategy, an estimation of the query response time is presented, giving an indication of the query processing performance. In the sequel we denote with $Cost_{\text{act}}$ the cost to activate a database, with $Cost_{\text{db}}$ the processing cost in each database, and with $Cost_{\text{result}}$ the cost to collect the results from a database. We assume that the network does not support multicasting. In a different case, the derived costs will be slightly different.

Concurrent processing
A message comprising of the query vector and the number k of nearest neighbors requested is submitted from the primary server to all source databases, one at a time. This costs:

$$Cost_{\text{act}} = Cost_{\text{trans}}(S_v + S_n)$$

Since all source databases receive the query request almost at the same time, the local processing cost equals:

$$Cost_{\text{db}} = \left(\text{INA}(k) + \frac{S_{\text{o}}}{S_{\text{p}}} \cdot k\right) \cdot T_p$$

Finally, the primary server must collect k objects from each source database. Therefore:

$$Cost_{\text{result}} = Cost_{\text{trans}}(k \cdot (S_n + S_{\text{o}}))$$

Summing up all costs we get:

$$\boxed{R_{CP}(k) = m \cdot Cost_{\text{act}} + Cost_{\text{db}} + m \cdot Cost_{\text{result}}} \tag{3}$$

Selective processing
All source databases are activated by sending the query vector and the number k of nearest neighbors requested. This costs:

$$Cost_{\text{act}} = Cost_{\text{trans}}(S_v + S_n)$$

For each subsequent source database (except the first one) the primary server must transfer the current k best distances:

$$Cost_{\text{act2}} = Cost_{\text{trans}}(k \cdot S_n)$$

because we must send the k distances of the best objects obtained so far. Let each source database j process C_j objects. Then, the local processing cost equals:

$$Cost_{\text{db}} = \left(\text{INA}(k) + \frac{S_{\text{o}}}{S_{\text{p}}} \cdot C_j\right) \cdot T_{\text{p}}$$

The transmission of C_j objects from source database j to the primary server costs:

$$Cost_{\text{result}} = Cost_{\text{trans}}((S_{\text{o}} + S_n) \cdot C_j)$$

Summing up all costs we get:

$$\boxed{R_{SP}(k) = m \cdot Cost_{\text{act}} + (m-1) \cdot Cost_{\text{act2}} + \sum_{j=1}^{m} Cost_{\text{db}} + \sum_{j=1}^{m} Cost_{\text{result}}} \tag{4}$$

Two-phase processing
First, the f source databases are activated by sending the query vector and the number k of nearest neighbors requested. This costs:

$$Cost_{\text{act}} = Cost_{\text{trans}}(S_v + S_n)$$

Each of the f source databases will process k objects in parallel, requiring cost:

$$Cost_{\text{db1}} = \left(\text{INA}(k) + \frac{S_{\text{o}}}{S_{\text{p}}} \cdot k\right) \cdot T_{\text{p}}$$

The transfer of k objects from each of the f source databases require cost:

$$Cost_{\text{result1}} = Cost_{\text{trans}}(k \cdot (S_{\text{o}} + S_n))$$

The activation of the rest $m - f$ source databases require the transfer of the current best k distances plus the query vector:

$$Cost_{\text{act2}} = Cost_{\text{trans}}(k \cdot S_n)$$

The $m - f$ source databases process C objects each. Therefore, the local processing cost is:

$$Cost_{\text{db2}} = \left(\text{INA}(k) + \frac{S_{\text{o}}}{S_{\text{p}}} \cdot C\right) \cdot T_{\text{p}}$$

The primary server must collect C objects from each source database (among the $m - f$) and therefore:

$$Cost_{\text{result2}} = Cost_{\text{trans}}(C \cdot (S_{\text{o}} + S_n))$$

In conclusion, the total cost for this strategy is given by:

$$\boxed{R_{2PP}(k) = m \cdot Cost_{\text{act}} + Cost_{\text{db1}} + f \cdot Cost_{\text{result1}} + (m - f) \cdot Cost_{\text{act2}}} \\ \boxed{+ Cost_{\text{db2}} + (m - f) \cdot Cost_{\text{result2}}} \quad (5)$$

Probabilistic processing
A message comprising of the query vector and the number $k/m + 1$ of nearest neighbors requested is submitted from the primary server to all source databases. This costs:

$$Cost_{\text{act}} = Cost_{\text{trans}}(S_v + S_n)$$

Since all source databases receive the query request almost at the same time, the local processing cost equals:

$$Cost_{\text{db}} = \left(\text{INA}\left(\frac{k}{m+1}\right) + \frac{S_{\text{o}}}{S_{\text{p}}} \cdot \left(\frac{k}{m+1}\right)\right) \cdot T_{\text{p}}$$

Subsequently, the primary server must collect $k/m+1$ objects from each source database. Therefore:

$$Cost_{\text{result}} = Cost_{\text{trans}}\left(\left(\frac{k}{m+1}\right) \cdot (S_n + S_{\text{o}})\right)$$

In the best case of **PRP** (**PRP**$_{\text{best}}$) no further processing is required. However, in a typical case (**PRP**$_{\text{avg}}$) let m' be the number of reactivated databases, where each one contributes with C_j objects. The reactivation cost per database equals the transmission cost of the best k distances determined so far:

$$Cost_{\text{act2}} = Cost_{\text{trans}}(k \cdot S_n)$$

Each of the reactivated databases will perform further processing in order to determine the best k matches. Therefore, the cost per database equals:

$$Cost_{\text{db2}} = \left(\text{INA}(k) + \frac{S_{\text{o}}}{S_{\text{p}}} \cdot C_j\right) \cdot T_{\text{p}}$$

Finally, each reactivated database will transmit C_j objects, with cost:

$$Cost_{\text{result2}} = Cost_{\text{trans}}((S_{\text{o}} + S_n) \cdot C_j)$$

Summing up we obtain:

$$\boxed{R_{PRP}(k) = m \cdot Cost_{\text{act}} + Cost_{\text{db}} + m \cdot Cost_{\text{result}} + m' \cdot Cost_{\text{act2}}} \\ \boxed{+ Cost_{\text{db2}} + m' \cdot Cost_{\text{result2}}} \quad (6)$$

It is evident that the performance of **CP** is quite predictable, since each source database processes and transmits exactly k objects. However, in order to predict the performance of **SP** and **2PP**, further analysis is required. We need the following lemmas[3] in order to proceed.

Lemma 3.2.1. *Assume that* $NC_j = \frac{k}{m}$ *for all* $1 \leq j \leq m$. *Then the following holds*:

1. *The first database we access will process and transmit k objects.*
2. *The n-th database* (*where* $n < m$) *we access, will process and transmit* $k - n \cdot \frac{k}{m}$ *objects in the worst case and* $\frac{k}{m}$ *objects in the best case.*
3. *The last* (*m-th*) *visited database will process and transmit exactly* $\frac{k}{m}$ *objects.*

Lemma 3.2.2. *The average number of objects processed and transmitted by a source database for a k-NN query by* **SP** *is*:

$$\bar{O}_{SP} = \left(\frac{m^2 + 5m - 2}{4m^2} \right) \cdot k$$

Lemma 3.2.3. *The average number of objects processed and transmitted by a source database for a k-NN query by* **2PP** *is*:

$$\bar{O}_{2PP} = \left(\frac{2mf + (m - f)(m - f + 1)}{2m^2} \right) \cdot k$$

Evidently, if each database contributes with exactly k/m objects, the **PRP** method needs only one phase, since no database will be reactivated. However, in a more typical case, some of the databases will be reactivated and further objects will be processed and transmitted. In such a case, the expected number of objects that each reactivated database will process is given by the following Lemma.

Lemma 3.2.4. *The average number of objects processed and transmitted by a source database for a k-NN query by the second step of* **PRP** *is*:

$$\bar{O}_{PRP} = \frac{k \cdot (m - 1) - m}{2 \cdot m}$$

According to the above lemmas the average execution time for each evaluation strategy is given by the following formulae:

$$\begin{aligned}
R_{CP}(k) &= m \cdot Cost_{\text{trans}}(S_v + S_n) + \left(\text{INA}(k) + \frac{S_\text{o}}{S_\text{p}} \cdot k \right) \cdot T_\text{p} \\
&\quad + m \cdot Cost_{\text{trans}}(k \cdot (S_n + S_\text{o})) \qquad (7) \\
R_{SP}(k) &= m \cdot Cost_{\text{trans}}(S_v + S_n) + (m - 1) \cdot Cost_{\text{trans}}(k \cdot S_n) \\
&\quad + m \cdot \left(\text{INA}(k) + \frac{S_o}{S_p} \cdot \bar{O}_{SP} \right) \cdot T_\text{p} + m \cdot Cost_{\text{trans}}((S_\text{o} + S_n) \cdot \bar{O}_{SP}) \qquad (8) \\
R_{2PP}(k) &= m \cdot Cost_{\text{trans}}(S_v + S_n) + 2 \cdot \left(\text{INA}(k) + \frac{S_\text{o}}{S_\text{p}} \cdot \bar{O}_{2PP} \right) \cdot T_\text{p} \\
&\quad + m \cdot Cost_{\text{trans}}(\bar{O}_{2PP} \cdot (S_\text{o} + S_n)) + (m - f) \cdot Cost_{\text{trans}}(k \cdot S_n) \qquad (9) \\
R_{PRP}(k) &= m \cdot Cost_{\text{trans}}(S_v + S_n) + \left(\text{INA}(k/m + 1) + \frac{S_\text{o}}{S_\text{p}} \cdot (k/m + 1) \right) \cdot T_\text{p} \\
&\quad + m \cdot Cost_{\text{trans}}((k/m + 1) \cdot (S_n + S_o)) + (m/2) \cdot Cost_{\text{trans}}(S_n \cdot k) \\
&\quad + \left(\text{INA}(k) + \frac{S_\text{o}}{S_\text{p}} \cdot \bar{O}_{PRP} \right) \cdot T_\text{p} + (m/2) \cdot Cost_{\text{trans}}(\bar{O}_{PRP} \cdot (S_\text{o} + S_n)) \\
& \qquad (10)
\end{aligned}$$

The scenario assumed in the above analysis (scenario A) is that the detailed object description is transmitted in addition to the distance from the query point. This is useful when the user requires the first answers to be available as quickly as possible, even if they do not correspond to the real nearest neighbors. As long as the size of each object is small (e.g. 100 bytes) there is relatively little overhead for processing and transmitting this extra information. On the other hand, for larger object sizes and large numbers of requested neighbors, this cost becomes very significant and may dominate with respect to the total response time. Therefore, another scenario (scenario B) that could be followed, is to first determine the object IDs and the distances to the query point, and then to reactivate the relevant databases in order to fetch the detailed description of only the best matches. Evidently, the cost for this last action is the same for every strategy. We do not present the equations for the second scenario, since are simpler versions of (7), (8), (9) and (10). However, in the analytical and experimental evaluation we demonstrate both cases.

Equations (7), (8), (9) and (10) give the expected execution time for each strategy when the system is lightly loaded, and therefore the waiting time is negligible. The behavior of the methods under system load is studied using experimental evaluation (see Section 5).

3.3. Analytical comparison

Summarizing the theoretical analysis, in this subsection we present a comparative study regarding the efficiency of the four strategies. We present some results, with respect to the formulae of the previous subsection, in order to study the behavior of the methods under different parameter values. The parameters modified and the corresponding values are summarized in Table 1. We note that these results correspond to the execution of a single query, which means that the impact of concurrent users is not taken into account.

In figure 2 the four query evaluation methods are compared, based on the analytic results. This figure includes the results for the case where the objects' detailed description is processed and transmitted. It is evident that the $\mathbf{PRP}_{\text{best}}$ method outperforms by factors the other candidates. The response time of all methods is increased by increasing the number of nearest neighbors (see figure 2(a)). Evidently **CP** is most affected by this increase, since every database processes and transmits k objects. Although **SP** transmits the smaller number of objects, the price paid is that no parallelism is exploited, and the response time is increased.

By increasing the number of dimensions, the processing cost in each database increases also. For large dimensionalities (e.g. 20) the cost to search the index becomes significant. In figure 2(b) it is observed that methods **2PP** and $\mathbf{PRP}_{\text{avg}}$ tend to converge, and the same is observed for methods **CP** and $\mathbf{PRP}_{\text{best}}$. For smaller dimensionalities ($<$10) the **PRP** methods show clearly the best performance.

The impact of the effective network speed on the performance of the methods is illustrated in figure 2(c). For small effective network speed (large network traffic), the **CP** shows the worst performance, since it transmits more objects than the other methods, and therefore the network becomes the bottleneck.

An interesting observation (see figure 2(d)) is that the performance of **SP** is affected in a negative manner by increasing the number of databases, whereas the response time of the

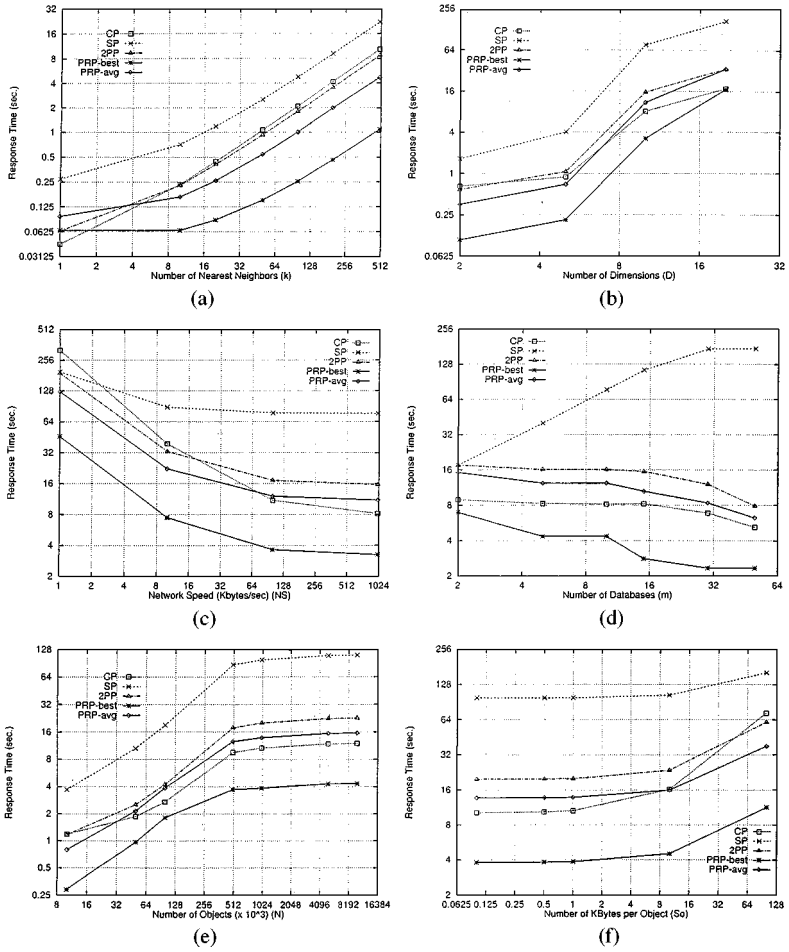

Figure 2. Performance of methods for scenario A (logarithmic scales). (a) $N = 1$ million, $m = 10$, $S_o = 1000$, $D = 2$, $NS = 1$ MByte/sec; (b) $N = 1$ million, $m = 10$, $S_o = 1000$, $k = 30$, $NS = 1$ MByte/sec; (c) $N = 1$ million, $m = 10$, $S_o = 1000$, $k = 30$, $D = 10$; (d) $N = 1$ million, $k = 30$, $S_o = 1000$, $D = 10$, $NS = 1$ MByte/sec; (e) $m = 10$, $k = 50$, $S_o = 1000$, $D = 10$, $NS = 1$ MByte/sec; and (f) $N = 1$ million, $k = 50$, $m = 10$, $D = 10$, $NS = 1$ MByte/sec.

other methods is reduced. The cause for this behavior is that **SP** does not exploit intraquery parallelism.

The increase in the number of objects is depicted in figure 2(e). Evidently, all methods are affected significantly. Finally, in figure 2(f), the response time with respect to the object size is illustrated. The impact on object size growth is stronger for **CP**, since it processes and transmits more objects than the other methods.

In figure 3 we illustrate the performance of the methods for the case where the detailed object description is not transmitted. It is observed that the results are not modified drastically with respect to the results in figure 2.

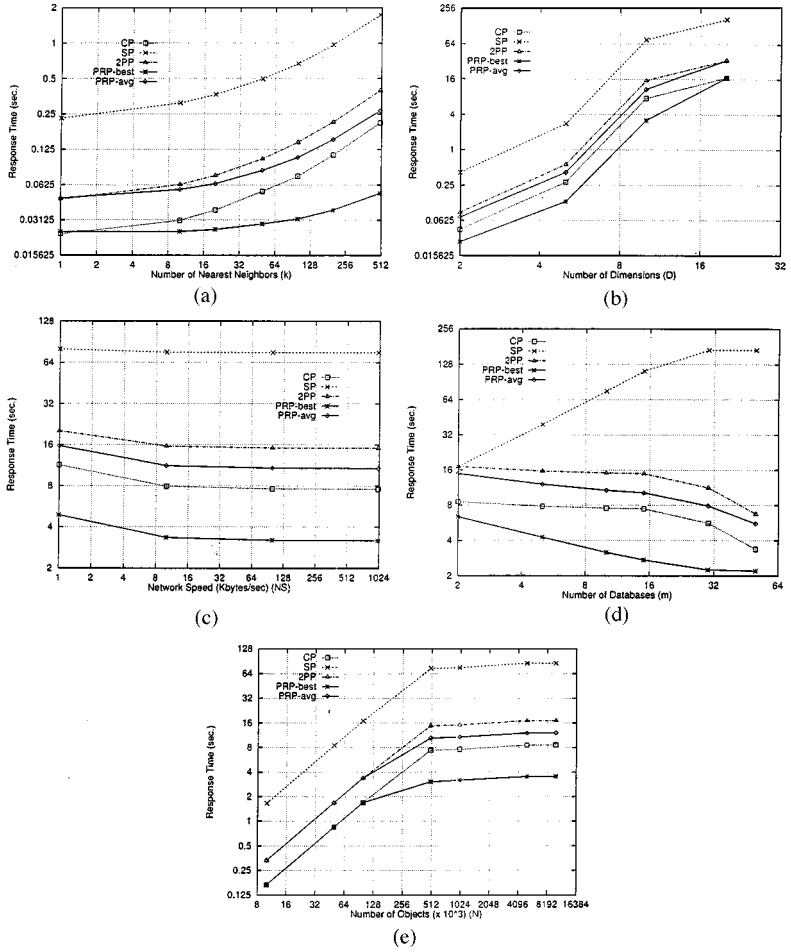

Figure 3. Performance of methods for scenario B (logarithmic scales). (a) $N = 1$ million, $m = 10$, $S_o = 1000$, $D = 2$, $NS = 1$ MByte/sec; (b) $N = 1$ million, $m = 10$, $S_o = 1000$, $k = 30$, $NS = 1$ MByte/sec; (c) $N = 1$ million, $m = 10$, $S_o = 1000$, $k = 30$, $D = 10$; (d) $N = 1$ million, $k = 30$, $S_o = 1000$, $D = 10$, $NS = 1$ MByte/sec; and (e) $m = 10$, $k = 50$, $S_o = 1000$, $D = 10$, $NS = 1$ MByte/sec.

The results presented in figures 2 and 3 correspond to a single user system, with no other interference. In a general case however, many users are posing queries to the database, resulting in network traffic and competition for the CPU in each database. For example, although the **SP** method does not support intraquery parallelism, supports interquery parallelism, because it is possible to access all m databases for m different queries. On the other hand, we expect a large performance degradation for **CP** method, since for large number of concurrent users queues will grow larger in disks, CPU and the network. In the next section we examine the impact of concurrent users, giving experimental results on a real implementation of the query evaluation strategies over a network of workstations.

4. The impact of derived data

In the previous section, we discussed evaluation strategies assuming that no derived data are available in the primary server. Therefore, all m source databases need to be visited in order to determine the best k matches to a given query object. However, in real applications, the presence of derived data is very important, in order to avoid searching large portions of the data space without a chance to retrieve relevant objects. Moreover, we may avoid visiting a particular source database, if we are absolutely sure that no relevant objects can be found, reducing network contention and saving overall system resources. Several types of derived data can be useful, ranging from simple numerical values (e.g. the number of objects in the database) to more sophisticated ones and difficult to obtain (e.g. an exact description of the object distribution). In this paper, we focus on derived data information that represent Minimum Bounding Boxes (MBB) of a set of objects. In other words, some descriptors are used to group objects in sets, e.g. two MBBs enclosing two different sets of objects.

In order to be able to discard quickly data space portions not related to the answer set, we require the presence of a set of MBBs stored in the primary server. For each source database j, the primary server maintains a number of MBBs. The smaller the overlap of these MBBs the better the discrimination during query processing. Also, a large number of MBBs helps the discrimination process.

To illustrate the use of MBBs for discrimination among objects, we present an example in figure 4. Two MBBs are shown, each holding 5 points in the 2-d space. Assume that the three nearest neighbors with respect to point Q_p are required. Let the circle enclose the best matches determined so far, namely, the points 1, 2 and 3 of MBB 1. Then we can safely avoid the search in MBB 2, since there is no intersection with the circle.

Consider now a query point and a number of MBBs. The question posed is which MBB are we going to visit first and how can we safely prune any portions of the data space that are not promising. The order that we access the MBBs (and consequently the source databases), has a major impact on the efficiency of a query processing strategy, since it is highly correlated to the number of transmitted objects. The following lemma (which is easily generalized for an arbitrary number of source databases) shows why a "good" visiting order of the source databases is necessary and also explains what "good" means.

Lemma 4.1. *Assume we have only two source databases SDB_1 and SDB_2 with net contributions NC_1 and NC_2 respectively, for a specific k-NN query. Assume further, without loss*

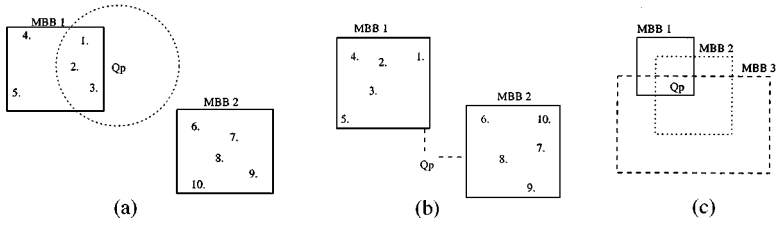

Figure 4. (a) Use of two MBBs for discrimination, (b) The nearest neighbor of Q_p is not in MBB 1, (c) A query point Q_p enclosed by many MBBs.

of generality, that $NC_1 \leq NC_2$. *Then, the sum of contributions,* $C_1 + C_2$, *is maximized if the source databases are accessed in increasing net contribution order* (*i.e.* SDB_1 *first and* SDB_2 *second*), *and is minimized if they are accessed in decreasing net contribution order* (*i.e.* SDB_2 *first and* SDB_1 *second*).

An approach used in [24] is to visit the MBBs according to the *MINDIST* distance. The $MINDIST(Q_p, R)$ distance is defined as the minimum distance between a query point Q_p and an MBB R. Therefore, a sorted list of MBBs with respect to the query point is formulated and then we investigate each MBB, following the order. There are two main drawbacks with this approach, illustrated in figure 4:

1. The fact that the query point Q_p is closer to MBB R does not provide any guarantee that also the nearest neighbor(s) of Q_p will be found in R.
2. By definition, if a query point Q_p falls inside an MBB R, then $MINDIST(Q_p, R) = 0$. Therefore, in the case where Q_p falls inside many MBBs $R_1, \ldots, R_n$, we are forced to select an MBB randomly.

Despite the above drawbacks of the *MINDIST* approach, the method is simple and easily implemented. In a separate section we discuss further improvement that requires additional information. In the following lines, the query evaluation strategies are presented taking into account the derived data information.

CP
1. *Determine the relevant source databases from derived data*.
2. *Send the query to the relevant databases*.
3. *Collect all answers*.
4. *Determine the best k matches*.

SP
1. *Determine the relevant source databases from derived data*.
2. *Using the MINDIST metric, find the best source database to access*.
3. *Send the currently best distances to the database*.
4. *Collect answers*.
5. *Discard any source databases that do not require access*.
6. *If there is no database to access then STOP else GOTO 2*.

2PP
1. *Determine the relevant source databases from derived data*.
2. *Using the MINDIST metric, find the best* f *databases to access*.
3. *Collect answers from the* f *databases*.
4. *Determine the currently best distances*.
5. *Discard any source databases that do not require access*.
6. *If there is no database to access then STOP*.
7. *Assume that s databases require access currently*.
8. *Access the s databases and collect the new answers*.
9. *Determine the best k matches*.

PRP
1. *Determine the relevant source databases from derived data*.
2. *Send the query to the r relevant databases, and collect $k/r + 1$ objects from each one*.
3. *Determine the current set of nearest neighbors*.
4. *Reactivate some of the databases if needed*.
5. *Determine the best k matches*.

In all methods, we need first to determine the relevant source databases, and to discard any databases that is impossible to contribute to the answer set. This is performed by means of the *MAXDIST* metric. The *MAXDIST* between a point and an MBB is defined as the distance from the point to the furthest vertex of the MBB. The following lemma explains:

Lemma 4.2. *Assume we have a set $\mathcal{M}_j$ of MBBs for each source database j. Let M^n_{ji} denote the number of objects that the MBB M_{ji} encloses. For simplicity let M^n_{ji} be equal for all j and i. We denote by R the distance between the query point Q_p and the $\lceil k/M^n_{ji} \rceil$-th MBB with respect to the MAXDIST metric, where k is the number of nearest neighbors requested. Then, all objects that participate in the answer set of nearest neighbors lie in the circle with center Q_p and radius R.*

5. Experimental study

5.1. Preliminaries

The performance evaluation of the processing strategies were carried out on a cluster of five Silicon Graphics workstations, comprising the source databases. We used a SUN Sparcstation-4 for the primary server. The workstations were interconnected via a 10 Mbit/sec Ethernet. Two types of processes were defined: 1) a client process running on the primary server and 2) a server process running on each source database. The responsibility of the client process is to pose queries to the source databases, whereas the responsibility of a server process is to serve the queries that are directed to the corresponding source database. The programs were coded in the C programming language under UNIX and the interprocess communication was based on the TCP/IP stream sockets programming interface [26].

We assume that each source database maintains an R-tree index [4, 15] for object storage and manipulation. Other data structures could have been used equally well. We generated random points in the 2-d, 3-d, 5-d and 10-d spaces. We can distinguish two ways to partition the objects to the source databases. In the first one, random assignment of objects to databases is used. In this approach, almost all source databases must be accessed in order to answer a similarity query. In the second one, each database is responsible for a small portion of the data space. In this approach, few databases must be accessed during query processing. Experiments have been conducted for the first case only, for brevity.

In order to study the performance of the methods under system load, we assume that users are posing queries concurrently to the primary server. Also, several values of the number

of nearest neighbors requested were used and different object sizes. For each experiment the average response time per similarity query was calculated. Each user poses 10 queries in total, and the queries are executed one-by-one.

5.2. Cost model evaluation

In a previous section a cost model has been derived for each query processing method. In order for these cost models to be useful, they should accurately predict the performance in real situations. Therefore, we start the experimental evaluation of the methods by first comparing the analytical formulae to the actual running time of each method.

In figure 5 the theoretical and measured response time for queries are depicted for each method. The parameters used for the evaluation are summarized below: $N = 100{,}000$, $NS = 1$ MByte/sec, $m = 5$, $D = 2$, $S_0 = 1000$ bytes, $f = 1$. The graphs are plotted in logarithmic scales in order for the differences to be more clear. It is evident that the cost models are quite accurate, since the maximum relative error is around 20%, whereas the average relative error is around 10%. Therefore, the cost model can be used to accurately predict the performance of a query evaluation method. This enables the use of the formulae for query optimization purposes or for selecting the appropriate method to answer a query according to the value of the parameters. More specifically, if one of the critical parameters (e.g. the effective network speed) changes, then by consulting the formulae the best method for the current settings can be selected. This gives the flexibility to the query execution engine to select the evaluation method that is expected to give the most promising results.

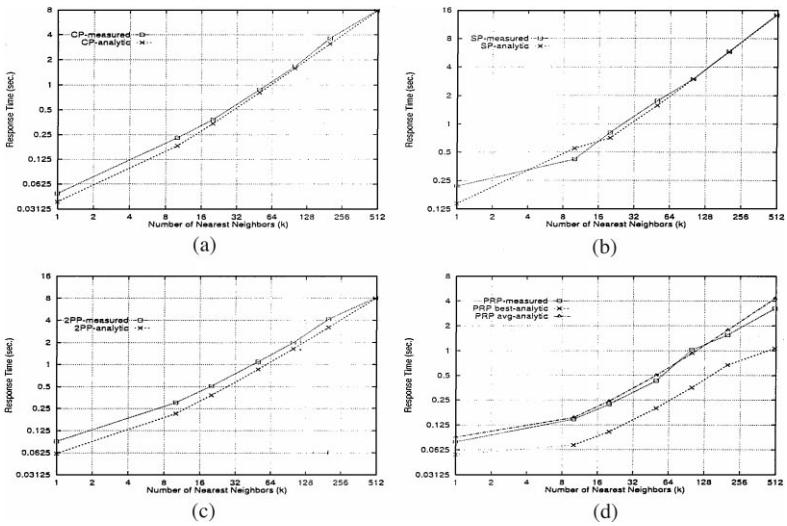

Figure 5. Cost model evaluation (logarithmic scales). (a) Concurrent Processing (CP), (b) Selective Processing (SP), (c) Two-Phase Processing (2PP), and (d) Probabilistic Processing (PRP).

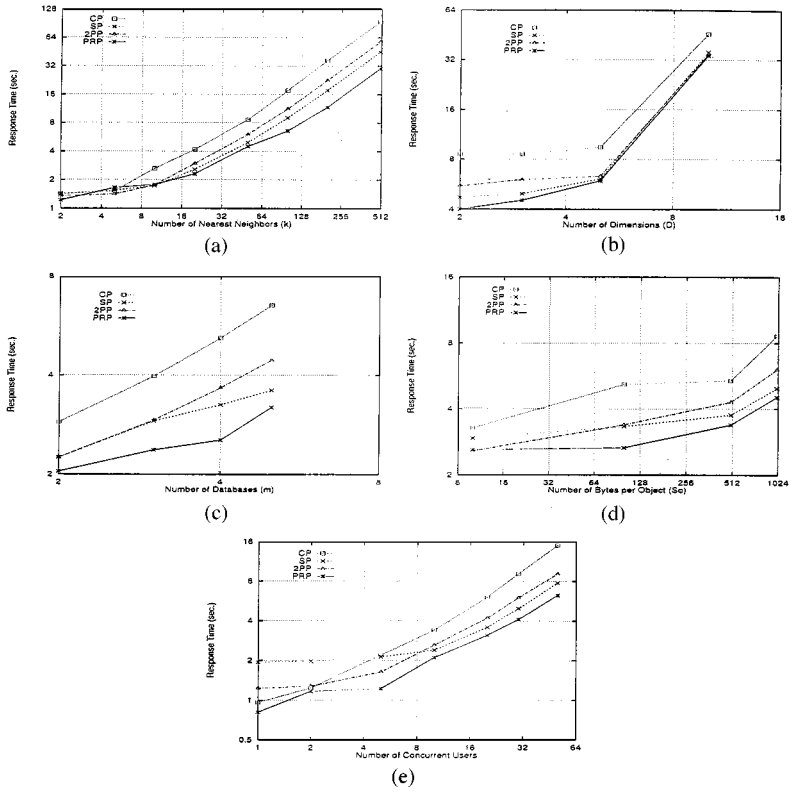

Figure 6. Measured response time for scenario A (logarithmic scales). (a) Variable k, (b) Variable D, (c) Variable m and N, (d) Variable S_o, and (e) Variable number of users.

5.3. Experimental results

In this subsection we illustrate representative results with respect to the real performance of the query evaluation strategies. Figure 6 illustrates the results when the detailed objects' description is processed and transmitted by the databases, whereas in figure 7 these costs are not included. All graphs are plotted in logarithmic scales. In order to investigate the performance of the methods under system load, we assume that users are posing queries concurrently. Each user submits queries to the primary server one-by-one. The response time illustrated in the graphs is the average response time per query, calculated over all users. We note that the cost includes CPU time, since for large number of users we expect this cost to be significant, because of waiting time.

In figure 6(a) we depict the response time with respect to the number of nearest neighbors k. For this experiment we used the following parameter values: $N = 250{,}000$, $D = 3$, $m = 5$, $S_o = 1000$, $f = 1$. Each database holds 50,000 objects. There are 30 users posing

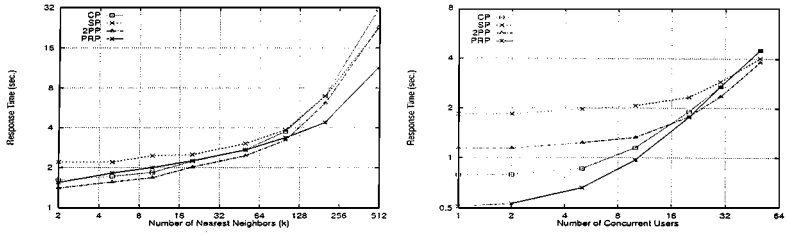

Figure 7. Measured response time for scenario B (logarithmic scales). (a) Variable k, and (b) Variable number of users.

queries concurrently. For a small number of k (e.g. 2,3), **CP** performs quite well. However, when k increases, the performance of **CP** degrades. The reason is that **CP** demands k objects from each activated database, resulting in high resource consumption in the CPU, the disk, and the network. An interesting observation is that although **SP** does not exploit intraquery parallelism, its performance is very good in a multiuser system. However, **PRP** shows the best performance.

Figure 6(b) illustrates the method performance for different number of dimensions. Each database holds 50,000 objects. The rest of the parameters have as follows and $k = 50$, $m = 5$, $S_o = 1000$, $f = 1$. There are 30 concurrent users posing queries. Evidently, all methods are affected drastically by increasing the space dimensionality. The reason is that CPU and disk costs are higher, due to the increased processing cost of the index in each database.

Figure 6(c) illustrates the method performance for different number of databases, and different number of objects. Each database holds 50,000 objects. The rest of the parameters have as follows and $k = 50$, $D = 10$, $S_o = 1000$, $f = 1$. There are 30 concurrent users posing queries. **PRP** demonstrates the best performance, whereas the performance of **CP** degrades. By increasing the number of databases, more network traffic is anticipated, since **CP** requests k objects from each database. Also, **SP** and **2PP** have similar performance.

The impact of the object size is illustrated in figure 6(e). This graph was produced using $N = 250{,}000$, $k = 50$, $D = 10$, $f = 1$ and assuming that there are 30 users posing queries. Evidently **CP** is affected more, and we expect higher degradation for larger number of bytes per object. Again **PRP** and **SP** demonstrate similar performance, and **PRP** performs the best.

The impact of the number of concurrent users is depicted in figure 6(d). Again, $N = 250{,}000$, $k = 50$, $D = 3$, $S_o = 1000$, and $f = 1$. When the number of users is relatively small (i.e. <10), the performance of **SP** degrades. This behavior is explained by taking into account that **SP** does not exploit intraquery parallelism. Therefore, the CPU and disk costs in each database are added, resulting in performance degradation. **CP**, **2PP** and **PRP** show similar performance. For a large number of concurrent users, **CP** is affected in a negative manner, because of bottlenecks. The other methods demonstrate similar performance, with **PRP** being the most efficient method.

In Figure 7 we illustrate the performance of the methods for scenario B, where the detailed object description is not transmitted by the databases before the k best matches have been

determined. It is interesting to note that in most cases **SP** does not perform well, unlike scenario A. The network traffic is reduced, and this is in favor for **CP**, **2PP** and **PRP**. The only exception occurs for a large number of concurrent users (50) where the performance of the methods tend to converge (see figure 7(b)).

6. Discussion

In this section we discuss some issues that are of major importance and can be considered for improvements in the future:

- Care should be taken when designing the derived data. If the number of derived data objects is large, then the primary server may become a bottleneck due to the increased CPU time required to process them. The reason is that a large number of MBBs helps in better pruning during query processing but, on the other hand, increases the required processing time. Therefore, it would be useful to maintain a separate data structure for the derived data in order to speed up processing.
- The generation of the derived data is very important. If the objects are manipulated by the source databases using a data structure based on Minimum Bounding Boxes (e.g. R-trees, R^+-trees) then we can use an intermediate level of the tree in order to extract the MBBs needed by the primary server (see for example [18]). On the other hand, if the corresponding data structures are not MBB based, then the MBBs should be generated artificially.
- As explained in Section 4, the *MINDIST* approach can lead to a not so efficient access order of the source databases. A number of additional reference points may help in better ordering of the source databases. For example, a reference point may be the center of a cluster of objects. Therefore, if a specific cluster center is closer to the query point than other cluster centers, we have a good chance that this particular cluster will contribute the most to the final answer set of nearest neighbors. The point here is that additional computation is required to extract the clustering information from the source databases and, also, to exploit this information during query processing. The fact that cluster centers will improve processing performance stills needs to be justified through experimental evaluation.
- A major issue that affects the performance of all methods is the placement of objects to databases. Since we allow each database to have separate and complete control over its stored objects, insertions and deletions of objects will create high overlaps among the data spaces of the databases. This effect results in accessing many source databases for a single query. On the other hand, if we force centralized control (i.e. a single site is responsible for insertions/deletions/reorganizations), then it is still an open problem to derive optimal data placement techniques for similarity query processing.

7. Concluding remarks

We have examined the problem of multidimensional similarity query processing in a distributed system. The problem is well studied for the centralized case, and a number of very

efficient methods have been proposed. However, in a distributed database system we have to take into consideration the communication overhead, in addition to the CPU and I/O cost, especially when the size of each object is not negligible. Four query evaluation strategies were developed and studied analytically and experimentally. The efficiency of each method depends heavily on several parameters such as the number of available source databases, the object placement in the databases, the object volume, the space dimensionality, the communication speed, the number of nearest neighbors requested, and the number of users issuing queries concurrently.

Each of the studied query evaluation strategies has its advantages and disadvantages, and the performance varies according to the parameters. Generally, methods **2PP** and **PRP** are the most robust, whereas **CP** and **SP** are sensitive to the multiprogramming degree and the database processing cost. However, they can be used in special cases. The developed cost model can be used in order to predict the performance of a query evaluation method.

In the near future we plan to implement the query evaluation methods using a larger number of workstations (>5), and also to implement and study algorithms for similarity search in a parallel machine based on a shared-nothing or shared-memory architecture. It would be interesting to study speed-up, size-up and scale-up issues in such an environment.

Appendix I

Here we describe the derivation for the local processing cost in a source database. This cost is composed of two components: (i) the cost to search the index and (ii) the cost to access the objects. From [9] the average number of R-tree node accesses (INA) for a window query is given by the following equation:

$$\mathrm{INA}(k) = \sum_{j=0}^{h-1} \frac{N}{C_{\mathrm{eff}}^{h-j}} \cdot \prod_{i=1}^{D} \left(q_{\mathrm{s}} + \left(\frac{C_{\mathrm{eff}}^{h-j}}{N} \right)^{1/D} \right) \tag{11}$$

where N is the number of objects, h is the height of the tree, D is the dimensionality of the space, C_{eff} is the average capacity of a node, and q_{s} is the size of the window in each dimension. The space is normalized to the unit hypercube.

In order to exploit the previous formula, we assume that the objects are uniformly distributed in the address space. Under this assumption, if k denotes the number of objects contained in a query volume $\mathrm{Vol}(Q)$, the following holds:

$$\frac{\mathrm{Vol}(Q)}{\mathrm{Vol}(\mathrm{Space})} = \frac{k}{N}$$

Therefore, if the query volume corresponds to a hyper-rectangle, the window size q_{s} equals: $q_{\mathrm{s}} = \sqrt[D]{\mathrm{Vol}(Q)} \Rightarrow q_{\mathrm{s}} = \sqrt[D]{\frac{k}{N}}$.

Substituting the value of q_{s} in Eq. (11), we obtain a formula to estimate the expected number of node accesses during the execution of a nearest neighbor query asking for the k

nearest neighbors.

$$\mathrm{INA}(k) = \sum_{j=0}^{h-1} \frac{N}{C_{\mathrm{eff}}^{h-j}} \cdot \prod_{i=1}^{D} \left(\sqrt[D]{\frac{k}{N}} + \left(\frac{C_{\mathrm{eff}}^{h-j}}{N} \right)^{1/D} \right)$$

In the number of index node accesses we have to add the additional pages that need to be retrieved in order to fetch the objects from the disk. To read k objects each having a size of S_{o} bytes each, we need to read $\frac{S_{\mathrm{o}}}{S_{\mathrm{p}}} \cdot k$ disk pages. Since each access costs T_{p} seconds, the total local processing cost of answering a nearest neighbor query in a source database equals:

$$Cost_{\mathrm{local}} = \left(\mathrm{INA}(k) + \frac{S_{\mathrm{o}}}{S_{\mathrm{p}}} \cdot k \right) \cdot T_{\mathrm{p}} \tag{12}$$

We would like to note that the above cost model does not include buffer management or boundary effects due to high dimensionality. In these cases, other models could have been used instead. However, we used Eq. (11) because of its simplicity, and because it can be used to model non-uniform distributions [9].

Appendix II

Lemma 3.2.1. *Assume that $NC_j = \frac{k}{m}$ for all $1 \leq j \leq m$. Then the following holds*:

1. *The first database we access will process and transmit k objects.*
2. *The n-th database ($n < m$) we access, will process and transmit $k - (n-1) \cdot \frac{k}{m}$ objects in the worst case and $\frac{k}{m}$ objects in the best case.*
3. *The last (m-th) visited database will process and transmit exactly $\frac{k}{m}$ objects.*

Proof: We examine each case separately:

1. This is straightforward, since no precomputed distances exist before the access of the first source database.
2. We know that the net contribution of each source database j is $NC_j = k/m$. This means that k/m is the minimum number of objects that each source database will process and transmit. To prove the upper bound, let us assume that the currently accessed database, transmits $l > (k-(n-1)\cdot\frac{k}{m})$ objects. This means that we have found $l-(k-(n-1)\cdot\frac{k}{m})$ objects in this database that are closer to the query point than some objects among the $(n-1)\cdot k/m$. Moreover, this fact tell us that the net contribution of one or more databases that were accessed previously is not k/m but lower, which contradicts our assumption that the net contribution of each source database is k/m. Therefore, the upper bound in the number of transmitted objects for the n-th accessed database is $k-(n-1)\cdot\frac{k}{m}$.
3. This is a special case of 2 above, setting $n = m$. □

Lemma 3.2.2. *The average number of objects processed and transmitted by a source database for a k-NN query by* **SP** *is*:

$$\bar{O}_{SP} = \left(\frac{m^2 + 5m - 2}{4m^2}\right) \cdot k$$

Proof: According to Lemma 3.2.1, the n-th visited database source database will process and transmits k/m objects at best and $k - (n-1) \cdot \frac{k}{m}$ object at worst. Therefore, on average we expect that $(k - (n-2) \cdot \frac{k}{m})/2$ objects will be processed and transmitted. Taking into consideration all source databases, we have that the average number of processed objects per source database equals:

$$\bar{O}_{SP} = \frac{k}{m} + \frac{1}{m} \cdot \sum_{n=2}^{m} \frac{k \cdot m - (n-2) \cdot k}{2m} \Rightarrow \bar{O}_{SP} = \frac{m^2 + 5m - 2}{4m^2} \cdot k \qquad \square$$

Lemma 3.2.3. *The average number of objects processed and transmitted by a source database for a k-NN query by* **2PP** *is*:

$$\bar{O}_{2PP} = \left(\frac{2mf + (m - f)(m - f + 1)}{2m^2}\right) \cdot k$$

Proof: The f first accessed source databases will process k objects each, resulting in a total of $f \cdot k$ objects. The rest $m - f$ databases will process at best k/m objects and at worst $k - f \cdot \frac{k}{m}$ objects and on average $(k - (f-1) \cdot \frac{k}{m})/2$ objects. Taking all source databases into consideration we get:

$$\begin{aligned} \bar{O}_{2PP} &= \frac{f \cdot k + (m - f) \cdot \frac{m \cdot k - (f-1) \cdot k}{2m}}{m} \Rightarrow \bar{O}_{2PP} \\ &= \left(\frac{2mf + (m - f)(m - f + 1)}{2m^2}\right) \cdot k \end{aligned} \qquad \square$$

Lemma 3.2.4. *The average number of objects processed and transmitted by a source database for a k-NN query by the second step of* **PRP** *is*:

$$\bar{O}_{PRP} = \frac{k \cdot (m-1) - m}{2 \cdot m}$$

Proof: In the first step, each database has transmitted $k/m + 1$ objects. Therefore, at least $k/m + 1$ best matches have been determined. In the second step, each database will transmit at least 0 and at most $k - (k/m + 1)$ objects. Therefore, the average number of objects equals $\frac{k - (k/m+1)}{2}$. $\square$

Lemma 4.1. *Assume we have only two source databases SDB_1 and SDB_2 with net contributins NC_1 and NC_2 respectively, for a specific k-NN query. Assume further, without loss*

of generality, that $NC_1 \leq NC_2$. Then, the sum of contributions, $C_1 + C_2$, is maximized if the source databases are accessed in increasing net contribution order (*i.e. SDB_1 first and SDB_2 second*), *and is minimized if they are accessed in decreasing net contribution order* (*i.e. SDB_2 first and SDB_1 second*).

Proof: Consider that we first visit SDB_1 and then SDB_2. The first database will contribute with k objects and the second with $NC_2 = k - NC_1$ objects (according to Lemma 3.2.1). This results in a total of $k + NC_2$ objects. Now, assume that we first access SDB_2 which will process k objects, and then SDB_1 which will process $NC_1 = k - NC_2$ objects. The total number of objects is $k + NC_1$. Evidently, $k + NC_1 \leq k + NC_2$ and this completes the proof. □

Lemma 4.2. *Assume we have a set $\mathcal{M}_j$ of MBBs for each source database j. Let M_{ji}^n denote the number of objects that the MBB M_{ij} encloses. For simplicity let M_{ji}^n be equal for all j and i. We donte by R the distance between the query point Q_p and the $\lceil k/M_{ji}^n \rceil$th MBB with respect to the MAXDIST metric, where k is the number of nearest neighbors requested. Then, all objects that participate in the answer set of nearest neighbors lie in the circle with center Q_p and radius R.*

Proof: The circle C contains at least k objects, since we select for the radius of the circle the MAXDIST to the $\lceil k/M_{ji}^n \rceil$-th MBB. If there is no other object in the circle, then the k found so far are the best k matches. Any other objects which is closer to the query point than any of k objects above, must lie in the circle necessarily. □

Acknowledgments

Work supported by the European Union's TMR program, CHOROCHRONOS (contract number FMRX-CT96-0056 (DG 12 - BDCN)).

Notes

1. The L_p metric between two n-d vectors x and y is defined as: $L_p(x, y) = (\sum_{j=1}^{n} |x_j - y_j|^p)^{1/p}$.
2. The details can be found in Appendix I.
3. The proofs of all lemmas are presented in Appendix II.

References

1. D.J. Abel, B.C. Ooi, K.-L. Tan, R. Power, and J.X. Yu, "Spatial join strategies in distributed spatial DBMS," in Proceedings of the 4th International Symposium in Spatial Databases (SSD '95), Portland, ME, USA, August 1995, pp. 348–367.
2. R. Agrawal, C. Faloutsos, and A. Swami, "Efficient similarity search in sequence databases," in Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms, 1993, pp. 69–84.
3. S. Arya, D.M. Mount, and O. Narayan, "Accounting for boundary effects in nearest neighbor searching," in Proceedings of the 11-th Annual Symposium on Computational Geometry, Vancouver, British Columbia, Canada, 1995, pp. 336–344.

4. N. Beckmann, H.P. Kriegel, and B. Seeger, "The R*-tree: An efficient and robust method for points and rectangles," in Proceedings of the ACM SIGMOD Conference, Atlantic City, NJ, 1990, pp. 322–331.
5. J.L. Bentley, B.W. Weide, and A.C. Yao, "Optimal expected-time algorithms for closest point problems," ACM Transactions on Mathematical Software, vol. 6, no. 4, pp. 563–580, 1980.
6. S. Berchtold, D. Keim, and H.-P. Kriegel, "The X-tree: An index structure for high-dimensional data," in Proceedings of the 22nd VLDB Conference, Bombay, India, 1996.
7. D. DeWitt and P. Valduriez, "Parallel database systems: The future of high performance database systems," Communications of the ACM, vol. 6, no. 6, pp. 85–98, 1992.
8. S.T. Dumais, "Latent semantic indexing (LSI) and TREC-2," in The 2nd Text Retrieval Conference, D.K. Harman (Ed.), MD, March 1994, pp. 105–115.
9. C. Faloutsos and I. Kamel, "Beyond uniformity and independence, analysis of R-trees using the concept of fractal dimension," in Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '94), Minneapolis, MN, USA, 1994, pp. 4–13.
10. C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," in Proceedings of the ACM SIGMOD Conference, Minneapolis, MN, USA, 1994, pp. 419–429.
11. C. Faloutsos and K.-L. Lin, "FastMap: A fast algorithm for indexing, data mining and visualization of traditional and multimedia datasets," in Proceedings of the ACM SIGMOD Conference, San Jose, CA, 1995, pp. 163–174.
12. R. Fagin, "Combining fuzzy information from multiple systems," in Proceedings of the 15-th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '96), Montreal, Canada, 1996, pp. 216–226.
13. J.H. Friedman, J.L. Bentley, and R.A. Finkel, "An algorithm for finding the best matches in logarithmic expected time," ACM Transactions on Mathematical Software, vol. 3, pp. 209–226, 1977.
14. R.H. Guting, "An introduction to spatial database systems," The VLDB Journal, vol. 3, no. 4, pp. 357–399, 1994.
15. A. Guttman, "R-trees: A dynamic index structure for spatial searching," in Proceedings of the ACM SIGMOD Conference, Boston, MA, 1984, pp. 47–57.
16. H.V. Jagadish, "Linear clustering of objects with multiple attributes," in Proceedings of the ACM SIGMOD Conference, Atlantic City, NJ, 1990, pp. 332–342.
17. H.V. Jagadish, "A retrieval technique for similar shapes," in Proceedings of the ACM SIGMOD Conference, Denver, CO, May 1991, pp. 208–217.
18. N. Koudas, C. Faloutsos, and I. Kamel, "Declustering spatial databases on a multi-computer architecture," in Proceedings of the Extending Database Technology Conference (EDBT '96), Avignon, France, 1996.
19. R. Laurini and D. Thompson, Fundamentals of Spatial Information Systems, Academic Press: London, 1992.
20. K. Lin, H.V. Jagadish, and C. Faloutsos, "The TV-tree: An index structure for high-dimensional data," The VLDB Journal, vol. 3, pp. 517–542, 1994.
21. A.N. Papadopoulos and Y. Manolopoulos, "Parallel processing of nearest neighbor queries in declustered spatial data," in Proceedings of the 5th ACM-GIS Workshop in Advances in Geographical Information Systems, Rockville, MD, USA, November 1996, pp. 35–43.
22. A.N. Papadopoulos and Y. Manolopoulos, "Performance of nearest neighbor queries in R-trees," in Proceedings of the 6th International Conference on Database Theory (ICDT '97), Delphi, Greece, January 1997, pp. 394–408.
23. A.N. Papadopoulos and Y. Manolopoulos, "Similarity query processing using disk arrays," in Proceedings of the ACM SIGMOD Conference, Seattle, Washington, USA, 1998, pp. 225–236.
24. N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in Proceedings of the ACM SIGMOD Conference, San Jose, CA, USA, 1995, pp. 71–79.
25. A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," IEEE Transactions on Systems, Man and Cybernetics, vol. smc-13, no. 3, pp. 353–362, 1983.
26. W.R. Stevens, UNIX Network Programming, Prentice-Hall, 1990.
27. P. Valduriez and T. Ozsu, Principles of Distributed Database Systems, Prentice Hall, 1991.
28. D. White and R. Jain, "Similarity indexing: Algorithms and performance," in Proceedings of the SPIE: Storage and Retrieval for Image and Video Databases IV, Jan Jose, CA, USA, 1996, vol. 2670, pp. 62–75.