

Multi-query Optimization for Distributed Similarity Query Processing

Yi Zhuang¹Qing Li²Lei Chen³¹College of Computer Science, Zhejiang University²Dept of Computer Science, City University of Hong Kong³Dept of Computer Science and Engineering, Hong Kong University of Science and Technology
zhuangyi@zju.edu.cn, itqli@cityu.edu.hk, leichen@cse.ust.hk

Abstract

This paper considers a multi-query optimization issue for distributed similarity query processing, which attempts to exploit the dependencies in the derivation of a query evaluation plan. To the best of our knowledge, this is the first work investigating a multi-query optimization technique for distributed similarity query processing (MDSQ). Four steps are incorporated in our MDSQ algorithm. First when a number of query requests (i.e., m query vectors and m radiuses) are simultaneously submitted by users, then a cost-based dynamic query scheduling (DQS) procedure is invoked to quickly and effectively identify the correlation among the query spheres (requests). After that, an index-based vector set reduction is performed at data node level in parallel. Finally, a refinement process of the candidate vectors is conducted to get the answer set. The proposed method includes a cost-based dynamic query scheduling, a Start-Distance (SD)-based load balancing scheme, and an index-based vector set reduction algorithm. The experimental results validate the efficiency and effectiveness of the algorithm in minimizing the response time and increasing the parallelism of I/O and CPU.

1 Introduction

With the development of Internet and multimedia technologies, many applications require an efficient high-dimensional access method to support large-scale content-based multimedia retrieval in a distributed environment. Among them, the study of query-intensive similarity-based query is increasingly receiving more attention by many researchers [1] recently. Although the study of multi-/high-dimensional indexing has been carried out extensively [2, 3, 4, 9, 10], they all focused on creating an index based on high-dimensional data itself without considering the query performance improvement from the perspective of *Users' Query Request* (UQR). In some important database applications, such as data mining [17], time series [18] and image retrieval [19], etc, a sequence of

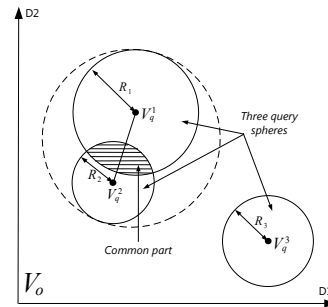


Figure 1. The intersection part of query spheres

interdependent queries may be posed simultaneously to the DBMS, especially in a distributed environment. The optimization of such sequences is called multi-query optimization [1], which attempts to exploit these dependencies in the derivation of a query evaluation plan. As shown in Figure 1, assume that users submit three UQRs such as Q_1 , Q_2 and Q_3 , there exist some possible correlations (*intersection*) between Q_1 and Q_2 , which means that they share the same search region (see the shaded part of Figure 1). It motivates us to further improve the query performance by exploiting the correlated query requests. That is, trying to combine the correlated query spheres together can enable faster answering the queries in a batch manner.

The state-of-the-art multi-/high-dimensional data indexing methods mentioned above all focus on a centralized [2] or distributed environment [8, 9, 10, 11]. Little research work has been done on the multi-query optimization of distributed similarity queries so far. In this paper, we propose *MDSQ* — an efficient multi-query optimization algorithm for distributed similarity query processing to significantly speed up the query performance in a distributed environment. The challenges of devising *MDSQ* include the following two main aspects:

- *Fast and effectively identifying the correlation of different query requests*: for many query requests, how to fast and effectively schedule the query requests is not a trivial issue.

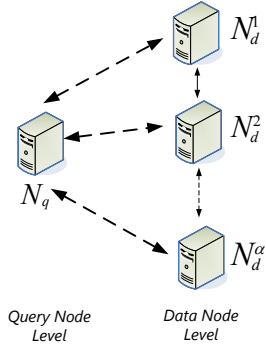


Figure 2. The topology of the distributed environment

- *Effective load balancing scheme*: for most of parallel database systems, it is critically important to develop an effective load balancing scheme which can maximize the query processing parallelism.

To address the above challenges, three enabling techniques are incorporated with the *MDSQ* scheme, which includes a *cost-based dynamic query scheduling*, a *Start Distance(SD)-based adaptive load balancing scheme* and an *index-based vector set reduction*. Figure 2 shows a framework of answering a *multiple similarity queries* in distributed environment. Assume that users submit a bunch of query requests (i.e., some query vectors and radius values) to a query node level N_q , the basic idea of the *MDSQ* is as follows: first, the cost-based dynamic query scheduling procedure is conducted to fast schedule query requests at the query node level. Then irrelevant high-dimensional vectors are fast filtered by using the *iDistance* [6] index in the data nodes in parallel, and meanwhile, the refinement (*distance computation*) process of the candidate vectors are conducted in parallel. Finally the answer set is sent back to the query node N_q , thereby completing the distributed high-dimensional range search. The aim of the paper is to minimize the network transfer latency and the I/O costs of *MDSQ*. We have implemented the *MDSQ* method and extensive experiments indicate that this method is specifically suitable for large high-dimensional data queries. Without loss of generality, Euclidean distance is used as the underlying distance function in this work. The contributions of this paper are summarized as follows:

- We propose a *MDSQ* as a novel multi-query optimization technique for distributed similarity query processing, to improve the efficiency of the distributed similarity queries in query-intensive applications.

- We introduce three such enabling techniques as cost-based dynamic query scheduling(*DQS*) scheme, *SD*-based load balancing scheme and an index-based vector set reduction(*VSR*), to progressively reduce the communication cost and maximize the parallelism in a distributed environment.

- Several extensive experimental studies are conducted to evaluate the efficiency, scalability and robustness of our proposed *MDSQ* algorithm.

The rest of the paper is organized as follows. Background is reviewed in Section 2. In Section 3, we formally define the problem and present some theoretical foundations for the proposed solutions. Then, in Section 4, we present three enabling techniques, *the cost-based dynamic query scheduling*, *the SD-based load balancing scheme* and *the index-based vector set reduction* to facilitate fast multi-query optimization for similarity queries over a distributed environment. In Section 5, the three proposed enabling techniques are seamlessly embedded in to a multi-query optimization algorithm. In Section 6, we perform comprehensive experiments to evaluate the effectiveness and efficiency of our proposed method; and we conclude the paper in Section 7.

2 Background

Our work is relevant to the domain of multi(*or* high)-dimensional indexing, existing works of which can be divided into two categories: *centralized high-dimensional indexing* [2] and *distributed high-dimensional indexing* [7,8,9].

Centralized multi(*or* high)-dimensional indexing has received extensive research attention in the past two decades [2]. In many of these methods, the data space is hierarchically divided into smaller regions, where data subspaces are overlap [3] or disjoint [4]. Other high-dimensional methods are proposed recently, such as vector-compression-based indexing (e.g., VA-file [5]) and distance-based indexing (e.g., *iDistance* [6]).

The indexing schemes mentioned above are proposed in a centralized mode. S. Berchtold *et al.* proposed a fast parallel similarity search in multimedia databases [7] by providing a near-optimal distribution of data items among the disks. Furthermore, in [8], a similarity search was presented using disk arrays. Recently, Peer-to-Peer(P2P)-based similarity search has been increasingly received attention. Among them, CAN [9] can be the first system supporting multi-dimensional data. Other system such as [11] used space filling curves to map multi-dimensional data to one dimensional space. pSearch [13], a P2P system based on CAN, is proposed for document retrieval in P2P networks. Another system also based on CAN is proposed by Sahin et al [12] in which the ranges are included into hash functions. However, exact search is highly inefficient in this system. SkipIndex [14] and [15] aim to support high dimensional similarity queries in the P2P network. The above distributed indexing schemes focus on the data without considering the query performance improvement from the perspective of query requests.

The study of multi-query optimization has also been extensively conducted in database field [1]. In general,

database systems frequently have to execute a set of related queries which may share several common sub expressions. Multi-query optimization exploits this, by finding evaluation plans that share common results. The proposed techniques include elimination of common sub-expressions, re-ordering of query plans, and using materialized views, etc.

3 Preliminaries and Foundations

3.1 Problem Definition

DEFINITION 1. A distributed network(DN) is a graph which is composed of Nodes and Edges, formally denoted as $DN=(N,E)$, where N is a set of nodes, E refers to a set of edges representing the network bandwidths for data transferring.

DEFINITION 2. The nodes in DN, formally denoted as $N=N_q+N_d$, can be logically divided into two categories: the query node (N_q) and data nodes (N_d), where N_q consists of one node and N_d is composed of α data nodes (N_d^i), where N_d^i refers to the i -th data node, for $i \in [1, \alpha]$ (cf. Fig. 2).

The list of symbols used throughout the rest of this paper is summarized and given in Table 1.

TABLE 1: Meaning of Symbols Used

Symbols	Meaning
Ω	a set of vectors
V_i	the i -th vector and $V_i \in \Omega$
D	the number of dimensions
n	the number of vectors in Ω
V_q^j	the j -th query vector user submits
$\Theta(V_q^j, r_j)$	the j -th query sphere, $j \in [1, m]$
$\Theta(V_q^j, R_j)$	the j -th query cluster, $j \in [1, m']$ and $m' < m$
$d(V_i, V_j)$	the distance between two vectors
Ω'	the candidate vector set
Ω''	the answer vector set
α	the number of the data nodes
$Vol(\bullet)$	the volume of \bullet

As defined in Definition 2, in a DN, a query node is responsible for submitting users' queries and receiving the answer set(Ω''); additionally, it can be used to schedule the query requests. The α data nodes are used to store the high-dimensional vectors(Ω) and their corresponding indexes; moreover, the vector set reduction and the refinement (*distance computation*) processes can be performed at the data node level in parallel, which can return the candidate vectors (Ω') and the answer set (Ω''), respectively.

This paper aims at solving the following problem:

Problem 1. Given m query requests submitted to the distributed network as defined in Definition 1, how can the m query requests be answered with the minimal network latency?

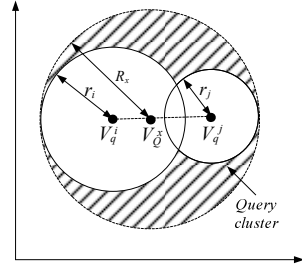


Figure 3. The MBS of the two spheres

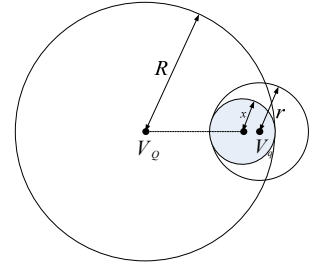


Figure 4. The MITS of two intersected spheres

3.2 Foundations

With loss of generality, assume that the vectors are uniformly distributed in high-dimensional spaces. For easy illustration, we use *sphere* to denote *hypersphere* in the rest of the paper.

DEFINITION 3. Given m query spheres: $\Theta(V_q^j, r_j)$, their corresponding minimal bounded sphere(MBS) can be formally denoted as: $MBS(\bigcup_{j=1}^m \Theta(V_q^j, r_j)) \supseteq \bigcup_{j=1}^m \Theta(V_q^j, r_j)$, such that $Vol(MBS(\bigcup_{j=1}^m \Theta(V_q^j, r_j)))$ is minimal.

Given two query sphere as in Figure 3: $\Theta(V_q^i, r_i)$ and $\Theta(V_q^j, r_j)$, their corresponding MBS is also a sphere which can be denoted as $MBS(\Theta(V_q^i, r_i), \Theta(V_q^j, r_j)) = \Theta(V_Q^x, R_x)$. The centre and the radius of $\Theta(V_Q^x, R_x)$ are derived by Theorem 1.

THEOREM 1. Given two query spheres: $\Theta(V_q^i, r_i)$ and $\Theta(V_q^j, r_j)$, the centre(V_Q^x) of their corresponding MBS

can be derived as: $V_Q^x = \frac{d(V_q^i, V_q^j) - r_i + r_j}{2 \times d(V_q^i, V_q^j)} \times (V_q^j - V_q^i) + V_q^i$ or

$V_Q^x = \frac{d(V_q^i, V_q^j) - r_j + r_i}{2 \times d(V_q^i, V_q^j)} \times (V_q^i - V_q^j) + V_q^j$, and the radius R_x as:

$$R_x = \frac{d(V_q^i, V_q^j) + r_i + r_j}{2}$$

PROOF. As shown in Figure 3, since $2R_x \geq d(V_q^i, V_q^j) + r_i + r_j$, then $Vol(MBS(\Theta(V_q^i, r_i), \Theta(V_q^j, r_j)))$ is minimal if $R_x =$

$$\frac{d(V_q^i, V_q^j) + r_i + r_j}{2}.$$

For the centre(V_Q^x), it can be derived as follows:

$$\begin{aligned} V_Q^x &= \frac{R_x - r_i}{d(V_q^i, V_q^j)} \times (V_q^j - V_q^i) + V_q^i \\ &= \frac{d(V_q^i, V_q^j) - r_i + r_j}{2 \times d(V_q^i, V_q^j)} \times (V_q^j - V_q^i) + V_q^i \end{aligned}$$

Similarly, V_Q^x of the MBS can also be represented as follows:

$$V_Q^x = \frac{d(V_q^i, V_q^j) - r_j + r_i}{2 \times d(V_q^i, V_q^j)} \times (V_q^i - V_q^j) + V_q^j \quad \square$$

DEFINITION 4. Given two spheres: $\Theta(V_Q, R)$ and $\Theta(V_q, r)$, their corresponding maximal inner tangent sphere

(MITS) can be a sphere $\odot(V_x, R_x)$ which is contained by the intersection part of the two spheres such that

$$R_x = \frac{r-d(V_\varrho, V_q)+R}{2}, \quad V_x = \frac{d(V_\varrho, V_q)-R+r}{2 \times d(V_\varrho, V_q)} \times (V_\varrho - V_q) + V_q \text{ or}$$

$$V_x = \frac{d(V_\varrho, V_q)+R-r}{2 \times d(V_\varrho, V_q)} \times (V_q - V_\varrho) + V_\varrho$$

As shown in Figure 5, given two spheres and their corresponding MBS, the effectiveness of clustering the two spheres can be guaranteed if the number of vectors in the two spheres is larger than half number of the vectors in the MBS at least. In other words, the volume of these two spheres should be larger than that of their corresponding MBS, which is described in Theorem 2.

THEOREM 2. Given two query spheres: $\odot(V_q^i, r_i)$ and $\odot(V_q^j, r_j)$, let their MBS be denoted as $\odot(V_\varrho^s, R_s)$, in order

to get $\frac{Vol(\odot(V_q^i, r_i) \cup \odot(V_q^j, r_j))}{Vol(\odot(V_\varrho^s, R_s))} > \frac{1}{2}$, the following condition (i.e., Eq.(1)) should be satisfied.

$$\begin{cases} 2 \times (r_i^D + r_j^D) > d(V_q^i, V_q^j)^D & \text{if } r_i + r_j > d(V_q^i, V_q^j) > r_i - r_j \\ 2 \times (r_i^D + r_j^D) > (r_i + r_j)^D & \text{if } r_i + r_j \leq d(V_q^i, V_q^j) \end{cases} \quad (1)$$

PROOF. For easy illustration, let A be $\odot(V_q^i, r_i)$, B be $\odot(V_q^j, r_j)$ and C be $\odot(V_\varrho^s, R_s)$, $r_i > r_j$. In Figure 5, there are three cases in terms of the distance between the two centres and the sum of the two radii.

Case 1: B is contained by A. In this case, as shown in Figure 5(a), we have: $r_i \geq d(V_q^i, V_q^j) + r_j$, then

$$\frac{Vol(A \cup B)}{Vol(C)} = \frac{Vol(A)}{Vol(A)} = 1 > \frac{1}{2} \quad (2)$$

Case 2: B intersects with A. In this case, as shown in Figure 5(b), we have: $r_i + r_j > d(V_q^i, V_q^j) > r_i - r_j$, then

$$\frac{Vol(A \cup B)}{Vol(C)} = \frac{Vol(A) + Vol(B) - Vol(A \cap B)}{Vol(C)} > \frac{1}{2} \quad (3)$$

As can be seen from Eq.(3), it is not trivial to exactly get the volume of $A \cap B$, so we use the volume of the MITS of A and B to approximate the volume of $A \cap B$. In Figure 4, given two intersected spheres: $\odot(V_\varrho, R)$ and $\odot(V_q, r)$, their corresponding MITS can be represented by a shaded circle whose radius x can be derived as: $x = \frac{r-d(V_\varrho, V_q)+R}{2}$. Then we have $Vol(\odot(V_\varrho, R) \cap \odot(V_q, r)) \approx Vol(\text{MITS}(\odot(V_\varrho, R), \odot(V_q, r))) = \frac{x^D \times \pi^{D/2}}{\Gamma(D/2+1)} = \frac{(r-d(V_\varrho, V_q)+R)^D}{\Gamma(D/2+1)} \times \pi^{D/2}$.

$$\text{MITS}(\odot(V_\varrho, R), \odot(V_q, r)) = \frac{x^D \times \pi^{D/2}}{\Gamma(D/2+1)} = \frac{(r-d(V_\varrho, V_q)+R)^D}{\Gamma(D/2+1)} \times \pi^{D/2}$$

Since $\frac{Vol(A \cup B)}{Vol(C)} = \frac{Vol(A) + Vol(B) - Vol(A \cap B)}{Vol(C)} > \frac{1}{2}$, then

$$\frac{\frac{r_i^D \times \pi^{D/2}}{\Gamma(D/2+1)} + \frac{r_j^D \times \pi^{D/2}}{\Gamma(D/2+1)} - Vol(A \cap B)}{\frac{(r_i + r_j + d(V_q^i, V_q^j))^D}{2} \times \pi^{D/2}} > \frac{1}{2} \quad (4)$$

Moreover, we have $Vol(\text{MITS}(A, B)) \approx Vol(A \cap B)$, so

$$Vol(A \cap B) \approx \frac{\left(\frac{r_i + r_j + d(V_q^i, V_q^j)}{2}\right)^D \times \pi^{D/2}}{\Gamma(D/2+1)} \quad (5)$$

Combing Eq.(4) with Eq.(5), we have

$$r_i^D + r_j^D - \frac{1}{2} \times \left(\frac{r_i + r_j + d(V_q^i, V_q^j)}{2}\right)^D > \left(\frac{r_i + r_j - d(V_q^i, V_q^j)}{2}\right)^D \quad (6)$$

Since $r_i + r_j > d(V_q^i, V_q^j)$, then we have

$$\frac{1}{2} \times \left(\frac{r_i + r_j + d(V_q^i, V_q^j)}{2}\right)^D > \frac{1}{2} \times d(V_q^i, V_q^j)^D$$

Additionally, $\left(\frac{r_i + r_j - d(V_q^i, V_q^j)}{2}\right)^D > 0$, therefore we have

$$2 \times (r_i^D + r_j^D) > d(V_q^i, V_q^j)^D.$$

Case 3: B tangent with A or B dose not intersect with A, In this case, as shown in Figure 5(c), we have: $r_i + r_j \leq d(V_q^i, V_q^j)$,

$$\frac{Vol(A \cup B)}{Vol(C)} = \frac{Vol(A) + Vol(B)}{Vol(C)} > \frac{1}{2}$$

$$\frac{\frac{r_i^D \times \pi^{D/2}}{\Gamma(D/2+1)} + \frac{r_j^D \times \pi^{D/2}}{\Gamma(D/2+1)}}{\frac{\left(\frac{r_i + r_j + d(V_q^i, V_q^j)}{2}\right)^D \times \pi^{D/2}}{\Gamma(D/2+1)}} > \frac{1}{2}$$

$$2 \times (r_i^D + r_j^D) > \left(\frac{r_i + r_j + d(V_q^i, V_q^j)}{2}\right)^D$$

Since $r_i + r_j \leq d(V_q^i, V_q^j)$, we have

$$2 \times (r_i^D + r_j^D) > (r_i + r_j)^D \quad (7)$$

Based on the above analysis, we can conclude that

$$\begin{cases} 2 \times (r_i^D + r_j^D) > d(V_q^i, V_q^j)^D & \text{if } r_i + r_j > d(V_q^i, V_q^j) > r_i - r_j \\ 2 \times (r_i^D + r_j^D) > (r_i + r_j)^D & \text{if } r_i + r_j \leq d(V_q^i, V_q^j) \end{cases} \quad \square$$

4 Enabling Techniques

To facilitate efficient multi-query optimization for similarity query(MDSQ) processing in a distributed environment, in this section, we introduce three enabling techniques of MDSQ including *cost-based dynamic query scheduling*, *adaptive load balancing scheme* and *index-based vector set reduction*, respectively. The purposes of these techniques are to minimize the transfer cost and maximize the query parallelism.

4.1 Cost-Based Dynamic Query Scheduling

As mentioned before, for several query requests, it is a non-trivial task to give an efficient order to such query request sequence. To address this problem, we propose a *cost-based Dynamic Query Scheduling* scheme, called DQS, to obtain new query clusters by using a cost-based clustering algorithm.

Specifically, the query request set can be formally denoted as:

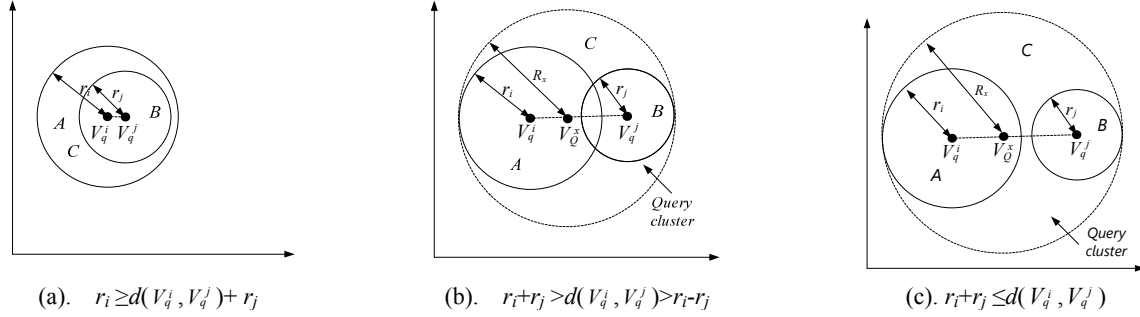


Figure 5. Three cases of query scheduling

$$QSet ::= \langle Q_1, Q_2, \dots, Q_m \rangle \quad (8)$$

where $Q_j = \Theta(V_q^j, r_j)$ refers to a query request submitted by the j -th user and $j \in [1, m]$.

The *DQS* algorithm is described below. Note that, line 3 in this algorithm is based on theorem 2. In other words, for any two spheres, if Eq.(1) is satisfied, then such two spheres can be combined together to obtain a new query sphere (*cluster*).

Algorithm 1. The query scheduling algorithm

Input: m query spheres;
Output: m' clustered query spheres;

1. **while**(TRUE)
2. **for** any two spheres **do**
3. **if** Eq.(1) is satisfied **then**
4. merge the two query sphere;
5. update the query sphere list;
6. $m \leftarrow m - 1$;
7. **end if**
8. **end for**
9. **end while** /* the value of m has been reduced to m' and $m' < m$ */
10. **return** m' (updated m) clustered query spheres;

4.2 SD-based Adaptive Load Balancing Scheme

To maximize the parallelism of vector set reduction at the data node level, we propose an adaptive *start-distance(SD)-based load balancing* scheme which can guarantee that the vectors can be equally distributed among the data nodes. More specifically, this method is to ensure that the cluster spheres in different data nodes can intersect with the query sphere. In other words, for each query, the vector set reduction(*VSR*) procedures to be studied in Section 4.3 can be performed in every data node in parallel.

DEFINITION 5(START DISTANCE). Given a vector V_i , the start distance of it is the distance between V_i and the origin V_o , denoted as $SD(V_i) = d(V_i, V_o)$, where $V_o = (0, 0, \dots, 0)$.

Before introducing the motivation of the *SD*-based vector allocation scheme, a high-dimensional space is first equally sliced into α “pieces” in terms of the start-distance, as shown in Figure 6. The scheme is based on the following key observations: if a cluster sphere does not intersect with the affected slices (i.e., the three grid slices in Figure 7) in which a query sphere is contained,

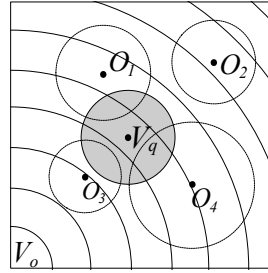


Figure 6. The slices in high-dimensional spaces

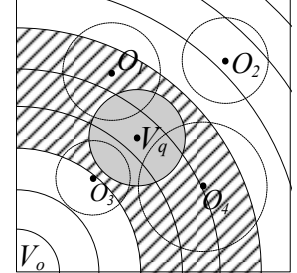


Figure 7. The three affected slices of the query sphere

then it is clear that the cluster sphere will not intersect with the query one. For the vector allocation in different data nodes(cf. Algorithm 2), the vectors in different data nodes are randomly selected from the vectors falling in the different slices, which means that the query sphere can intersect with the cluster spheres in almost every data node.

Algorithm 2. Vector allocation in data nodes

Input: Ω : the vector set, the α data nodes;
Output: $\Omega(1$ to $\alpha)$: the placed vectors in data nodes;

1. The high-dimensional space is equally divided into α slices in terms of start-distance;
2. **for** $j=1$ to α **do**
3. the $|n/\alpha|$ vectors($\Omega(j)$) are randomly selected from the each sub-range of start distance respectively;
4. $\Omega(j)$ is deployed in the j -th data node;
5. **end for**

4.3 Index-based Vector Set Reduction

As the vector dataset (Ω) is stored at the data node level, for a given query, it is obviously inefficient to directly conduct refinement process of the n vectors in Ω at the data node level. Therefore, in this section, we adopt the *iDistance* [6] in which a high-dimensional query is transformed to several range queries in single-dimensional space to support index-based vector set reduction. The purpose of it is to significantly reduce the computational cost in the refinement processing and the communication cost in distributed network. Algorithm 3 shows the *VSR* processing in the j -th data

node. The routine $RSearch(V_p, r)$ is the main range search algorithm which returns the candidate vectors of range search with centre V_q and radius r . Note that, the iDistance is also deployed at the data node level.

Algorithm 3. Vector set reduction ($VReduce(V_q, r, j)$)

Input: V_q : a query vector, r : a query radius,
 $\Omega(j)$: the vectors in the j -th data node and $j \in [1, \alpha]$,
Output: $\Omega'(j)$: the candidate vector set in the j -th data node

1. $S1 \leftarrow \Phi, S2 \leftarrow \Phi;$ /* initialization */
2. **for** each cluster sphere $\Theta(O_j, CR_j)$ **do**
3. **if** $\Theta(O_j, CR_j)$ **dose not intersect with** $\Theta(V_q, r)$ **then**
4. **break;**
5. **else**
6. $S2 \leftarrow RSearch(V_q, r);$
7. $S1 \leftarrow S1 \cup S2;$
8. **if** $\Theta(O_j, CR_j)$ **contains** $\Theta(V_q, r)$ **then** end loop;
9. **end if**
10. **end for**
11. **return** $S1;$ /* return candidate vectors */

5 The MDSQ Algorithm

Based on the three enabling techniques discussed above, in this section, we propose $MDSQ$ as a multi-query optimization algorithm for distributed similarity query processing. Before introducing this algorithm, we first assume that the vectors in Ω have been allocated at the data node level according to the SD -based load balancing scheme studied in Section 4.2. Then the allocated vectors in each data node are indexed by a iDistance [6], which is to support fast vector set reduction. The $MDSQ$ query processing algorithm is composed of three steps:

(1). **Dynamic Query Scheduling.** In the first step, m query requests(*spheres*) are submitted to the query node N_q , then the dynamic query scheduling(DQS) procedure is conducted to obtain m' query clusters as new query requests, where $m' < m$. The detailed steps of the DQS process is described in Algorithm 1.

(2). **Global Vector Set Reduction.** After the DQS processing at the query node level, the new m' query requests ($\Theta(V'_q, R_i)$) are packaged and sent to the α data nodes in parallel. Then, the irrelevant vectors in the α data nodes are filtered quickly by using the vector set reduction algorithm in parallel (cf. Section 4.3). Specifically, for each data node N_d^j , an input buffer called IB_j is created for caching the vectors in $\Omega(j)$, and an output buffer OB_j which is used to store the candidate vector set $\Omega'(j)$ is also created, where $j \in [1, \alpha]$. Once the candidate vector set $\Omega'(j)$ has been obtained, the refinement process(*distance computation*) of such candidate vectors is performed at the corresponding data node, as to be discussed next.

Algorithm 4. Global Vector Set Reduction($GVReduce(\Omega, m'$ query clusters)

Input: α : the number of data nodes,
 $\Omega(j)$: the sub vector set in the j -th data node,
 $\Theta(V'_q, R_i)$: the m' new query spheres,
Output: the candidate vector set $\Omega'(1$ to $\alpha)$

1. **for** $j:=1$ to α **do**
2. **for** $i:=1$ to m' **do**
3. $\Omega'(j) \leftarrow \Omega'(j) \cup VReduce(V'_q, R_i, j);$
4. **end for**
5. $\Omega'(j)$ is cached in the output buffer $OB_j;$
6. **end for**

Note that, in algorithm 4, the vector set reduction procedure (i.e., $VReduce(V_q, r, j)$) in the different data nodes is performed in parallel.

(3). **Refinement.** In the last step, the distances between the candidate vectors and the query vector V'_q are computed in the corresponding α data nodes in parallel. As detailed in Algorithm 5, if the distance is less than or equal to r_i , where $i \in [1, m]$, then the answer vector is sent to the query node N_q in the package-based manner. Different from the global vector set reduction procedure in which the m' query clusters are used, in the refinement step, the m query spheres are adopted to conduct the refinement processing, where $m' < m$.

Algorithm 5. Refinement ($Refine(\Omega', m$ query spheres))

Input: the candidate vector set $\Omega'(j)$ in the j -th data node,
 m query vectors V'_q and m radiuses r_i

Output: the answer vector set $\Omega''(1$ to $\alpha)$

1. **for** $j:=1$ to α **do**
2. $\Omega''(j) \leftarrow \Phi;$
3. **for** $i:=1$ to m **do**
4. **if** $d(V_i, V'_q) \leq r_i$ **and** $V_i \in \Omega'(j)$ **then**
5. $\Omega''(j) \leftarrow \Omega''(j) \cup V_i;$
6. **end if**
7. **end for**
8. $\Omega''(j)$ is cached in the output buffer $OB_j;$
9. **end for**

In the j -th data node N_d^j , an input buffer IB_j is set for the candidate vector set $\Omega'(j)$, and the memory M_j for the $\Omega'(j)$ is allocated in N_d^j , which is used to store the vectors of $\Omega'(j)$. Additionally, an output buffer OB_j is set, which is used to store the answer vectors temporarily. If the data size of the answer vector set in OB_j equals to the package size, then the answer vectors are sent to the query node N_q in the package-based transfer technique.

Algorithm 6 shows the detailed steps of our $MBSQ$ algorithm. It is worth mentioning that steps 4-6 are executed in parallel.

Algorithm 6. The $MDSQ$ Algorithm

Input: m query requests V'_q, r_j

Output: the query result Ω''

1. $\Omega' \leftarrow \Phi, \Omega'' \leftarrow \Phi;$ /* initialization */
2. m query requests are submitted to the query nodes $N_q;$
3. the dynamic query scheduling(DQS) is conducted at the query node level;
4. $\Omega' \leftarrow GVReduce(\Omega, m'$ query clusters);
5. $\Omega'' \leftarrow Refine(\Omega', m$ query spheres);
6. the answer vector set Ω'' is sent to the query node $N_q;$

6 Performance Evaluation

To verify the effectiveness and efficiency of our

proposed *MDSQ* method, in this section we conduct five groups of simulation experiments to demonstrate the performance of the *MDSQ* method. We have implemented the iDistance-based vector set reduction algorithm in C language and the iDistance is deployed at the data node level. B⁺-tree is used as a single-dimensional index structure and the index page size is set to 4096 byte. In our experiments, 100 users' query requests are randomly generated, and we run the following experiments 100 times to get the average response time. All experiments are run in a local network.

We use two data sets as experimental data: (i) the real life data, viz., the color histogram data from *UCI KDD Archive* [16], which includes 68040 32-dimensional color histogram features, the value range of each dimension is between 0 and 1; and (ii) the synthetic data, namely, 5,000,000 100-D vectors are randomly generated which are uniformly distributed, with each dimension's value ranging between 0 and 1.

6.1 Effect of VSR

In the first experiment, we study the effect of vector set reduction (*VSR*) on the performance of our *MBSQ*. Two methods are performed and compared for this purpose. Method 1 does not adopt the *VSR* algorithm: the refinement process of the vector set Ω is directly conducted at the data node level N_d without vector set reduction process, and the answer vector set Ω'' is then sent to the query node N_q . Method 2 uses the *VSR* algorithm: first the vector set Ω is reduced (via Algorithm 4) to the candidate vector set Ω' , then the refinement process of Ω' is further performed at the corresponding data node, finally the answer set Ω'' is sent to the query node N_q . As shown in Figure 8, with the increase of data size, the total response time using the *VSR* is superior to that of no *VSR*. This is because the irrelevant vectors which are filtered by the iDistance [6] index can dramatically reduce the computational cost in the refinement process and the network transfer cost.

6.2 Comparative Study on Speedup

In the following experiments, we make a comparative study on the speedup which is given below.

DEFINITION 6. Given m query requests, the speedup is the ratio of the total response time for m distributed similarity query (*DSQ*) to that of *MDSQ*, formally denoted as:

$$Speedup = \frac{\sum_{i=1}^m TIME_{DSQ}}{TIME_{MDSQ}} \quad (9)$$

6.2.1 Effect of DQS In this experiment, we study the effect of dynamic query scheduling (*DQS*) on the performance of the *MBSQ*. Two methods are performed and compared for this purpose. Method 1 does not adopt the *DQS* algorithm. Method 2 adopts the *DQS*

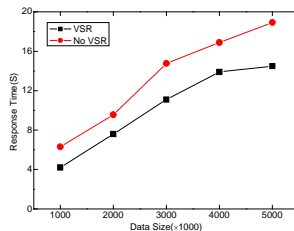


Figure 8. Effect of VSR

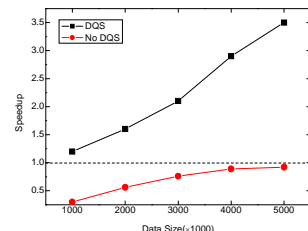


Figure 9. Effect of DQS

algorithm. (The *VSR* algorithm is adopted in both scenarios). Figure 9 shows that with the increase of data size, the speedup of Method 2 using the *DQS* is superior to that of no *DQS*. And the performance gap becomes larger since the multi-query performance can be significantly improved by using the *DQS* algorithm without compromising the network transfer cost.

6.2.2 Effect of Dimensionality To evaluate the effect of dimensionality on the speedup, we adopt the synthetic data set as the experimental data, in which the dimensionalities of these vectors range from 20 to 100 respectively. The data size is set to 1000,000 in the experiment. As shown in Figure 10, with the increase of dimensionality, the speedup keeps decreasing gradually since the vectors with higher dimensionality lead to higher CPU cost in the refinement (*distance computation*) processing. Moreover, for each answer vector, the increase of the dimensionality will result in a higher network transfer cost due to the larger data size of such vectors.

6.2.3 Effect of m In this experiment, we proceed to study the effect of the number of users' query requests (m) on the speedup by adopting the second data set. In this experiment, the value of m ranges from 20 to 100. Suppose that the data set size and the number of data nodes (α) are fixed, when m increases from 20 to 100, it is clear that the speedup as shown in Figure 11 is not vulnerable to m .

6.2.4 Effect of α In this experiment, we study the effect of the number of data nodes (α) on the speedup. Again, the second data set is used as the experimental data. As shown in Figure 12, with the increase of α , the speedup almost keeps increasing although the tendency of increase becomes slower after some numbers. When the number of the data nodes exceeds 40, the speedup is decreasing since the transfer cost from the query node to the data node level increases with the increase of data nodes, and it can counteract the parallelism of vector set reduction.

7 Conclusion

In this paper, we have presented *MDSQ* as the first piece of work (to the best of our knowledge) on multi-query optimization for distributed similarity query

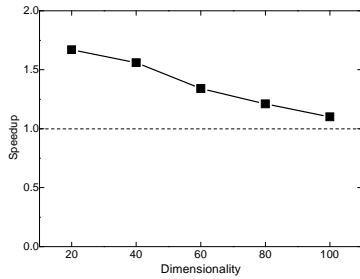


Figure 10. Effect of Dimensionality

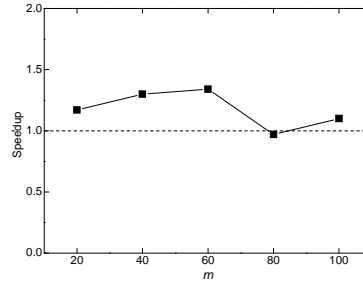


Figure 11. Effect of m

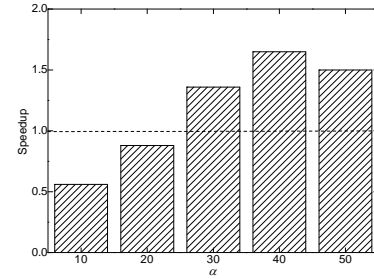


Figure 12. α vs. Speedup

processing, which specifically caters for the different bandwidth of nodes in the distributed network. To efficiently and effectively facilitate the *MDSQ*, we have proposed the three such enabling techniques as *the cost-based dynamic query scheduling scheme*, *the adaptive SD-based load balancing policy*, and *the index-based vector set reduction* to reduce the query response time. The experimental studies indicate that the proposed *MDSQ* method is especially suitable for handling distributed similarity queries over large-scale high-dimensional vector datasets, as it can achieve much better efficiency for the large-scale high-dimensional data in minimizing the communication cost and maximizing the parallelism in I/O and CPU.

Acknowledgement

This work is partially supported by a grant from the Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Science (reference number: IIP2006-1), the National Basic Research Program of China (973 Program) under grant No. 2006CB303000, and NSFC Project grant No. 60736013.

References

- [1] T.K. Sellis, Multi-query optimization, *TODS*, 13(1), 1988.
- [2] C. Böhm, S. Berchtold, D. Keim: Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases, *ACM Computing Surveys*, 33 (3), 2001.
- [3] A. Guttman, R-tree: A dynamic index structure for spatial searching. In *SIGMOD*, pp.47-54,1984.
- [4] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*, pp. 322-331. 1990.
- [5] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pp. 194-205,1998.
- [6] H.V. Jagadish, B.C. Ooi, K.L. Tan, C. Yu, R. Zhang: iDistance: An Adaptive B⁺-tree Based Indexing Method for Nearest Neighbor Search. *TODS*, 30(2), 364-397, June 2005.
- [7] S. Berchtold, C. Bohm, B. Braunmuller, D. A. Keim, etc. Fast Parallel Similarity Search in Multimedia Databases. In *SIGMOD*, pp. 1-12, 1997.
- [8] A.N. Papadopoulos, Y. Manolopoulos. Similarity query processing using disk arrays. In *SIGMOD*, pp. 225-236. 1998
- [9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM*, pp. 161-172, 2001.
- [10] J. Aspnes and G. Shah. Skip graphs. In *SODA*, pp 384-393, 2003.
- [11] J. Lee, H. Lee, S. Kang, S. Choe, and J. Song. CISS: An efficient object clustering framework for DHT-based peer-to-peer applications. In *DBISP2P*, pp.215-229, 2004.
- [12] O. D. Sahin, A. Gupta, D. Agrawal, and A. El Abbadi. A peer-to-peer framework for caching range queries. In *ICDE*, pp.165-176, 2004.
- [13] C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *SIGCOMM*, 2003.
- [14] C. Zhang, A. Krishnamurthy, and R. Y.Wang. Skip-index: towards a scalable peer-to-peer index service for high dimensional data. Technical report, Princeton University, 2004.
- [15] P. Kalnis, W.S. Ng, B.C. Ooi, K.L. Tan, Answering Similarity Queries in Peer-to-Peer Networks. *Information Systems*. 2006
- [16] *UCI KDD Archive*, <http://www.kdd.ics.uci.edu>, 2002
- [17] B. Braunmuller, M. Ester, H-P Kriegel, J. Sander, Multiple Similarity Queries: A Basic DBMS Operation for Mining in Metric Databases. *TKDE*, (13)1, pp. 79-95, 2001.
- [18] L. Xiang L. Chen, J. X. Yu, G. Wang and G. Yu, Similarity Match Over High Speed Time-Series Streams, in *ICDE*, 2007.
- [19] L. Xiang and L. Chen, Similarity Search in Arbitrary Subspaces under L_p-Norm, in *ICDE*, 2008.