Personalized query evaluation which returns top-k results according to users’ individual interests was once confined mainly to the web literature, while now it is increasingly becoming a factor in many other search systems, such as peer-to-peer systems, cloud computing systems etc. In this paper, we propose a novel method for efficient personalized query processing in P2P systems. The problem has never been discussed and remains imperative to be solved. By effectively calculating a corresponding search range on-the-fly which contains the dedicated top-k results based on the estimated multiscale data density in the current system, a personalized query is transformed into a special range query for efficient processing at run time. Singular value decomposition of the personalization matrix is deployed to simplify the process of range computation. Search request is then multicasted within the desired range, which optimizes the communication cost when fetching those results in the P2P network. Algorithms on multiscale density estimation, range computation and multicast routing are given as well as the theoretical analysis. Our extensive performance study confirms the effectiveness and efficiency of our method.

© 2012 Published by Elsevier Inc.

1. Introduction

Peer-to-Peer (P2P) systems derived from file sharing applications have been developed over a number of years. Effective and efficient search in P2P systems has always been regarded as a main consideration. Exact query [31,25,26], range query [16,1], preference query (for both quantitative evaluation, i.e., top-k query [35,30,34] and qualitative evaluation, i.e., skyline query [37,5]), similarity query [22,8] have been well studied in order to affiliate users’ search in P2P systems. Nevertheless, the most prevalent search form, say personalized search based on users’ individual interests (i.e., personalization) which has even broader application scenarios, such as social network search (e.g., Facebook [9]), scientific knowledge search (e.g., Scienccnet [28]), travel (e.g., P2PTravel [23]) and e-commerce (e.g., Tamago [32]), etc., has not been taken into account in previous studies. That is to say, they may not be effective or efficient for those queries containing different users’ interests (i.e., personalized queries which evaluate the results based on users’ individual interests).

Example 1. Let’s take the scientific knowledge search as an example. Suppose a user searches papers in terms of his or her personalization on attributes (e.g., authors, affiliations of authors, publishers of papers, topics of papers etc.). The desired top-k papers are evaluated and returned with respect to the user’s personalization on attributes. What should be noted is that not only the personalization on a single attribute (e.g. authors), but also the personalization on cross-attributes (e.g.,
authors and affiliations of authors) should be indicated by the user or learned by systems from the user’s search history to present his or her personalization precisely. This is for the purpose of removing the independency assumption between attributes (e.g., authors and topics of papers).

Personalized queries, which generally associate different importance (or weight) on attributes (or dimensions) in the score function and return the top-k results, stem from the web search literature [7], and originate in the centralized database for special cases, e.g., k-nearest neighbor search or preference search. With the same query evaluation mission and dilemma (i.e., avoiding users submerged in the millions of returned results and ranking the dedicated interest results precisely), personalized search in P2P systems needs to be solved imperatively. However, previous research is not directly applicable in P2P systems due to their distributed and dynamic nature. The difficulties to evaluate the personalized query in P2P systems lie in two main perspectives. Firstly, the score function varies from one user’s interest to another. The results (hence the search space) of a personalized query is subject to the personalization specified in the score function, while the personalization could be arbitrary. In highly dynamic P2P systems, it is very difficult and in fact impractical to adopt existing index methods used in centralized database systems to deal with preference queries, such as convex hull [3], Robust index [39], Threshold Algorithm [20] and so on, because of their extremely high index construction cost in distributed P2P systems. Secondly, P2P systems are highly dynamic since peers may join or leave frequently. Existing index methods suffer from a prohibitively high index maintenance cost. An effective personalized query evaluation method in P2P systems should be robust to the distributed and dynamic environment.

In this paper, we address the problem of personalized query evaluation in P2P systems, which returns the top-k results reflecting users’ individual personalization. It can be formally defined as follows: given a P2P system consisting of n peers, the task is to find the top-k data tuples which are ranked highest according to the score function reflecting the user’s personalization on the attributes of data tuples. Hereby, personalization on an attribute is referred to as a weight reflecting a user’s interest level on that attribute. If a user prefers one attribute to others, he or she can provide a higher weight on this attribute in the score function, resulting in more notable importance of the score on this attribute value. More generally, cross-attribute weights can also be specified, indicating inter-dependency among different attributes. Furthermore, it should be noted our personalized query evaluation algorithm is applicable for many P2P overlays, such as Content-addressable Network (i.e., CAN [25]), Ring-based networks (i.e., Chord [31], Pastry [26], and so on).

To cater for different users’ interests in P2P search, we present a novel method towards efficient personalized query evaluation in P2P systems. From the perspective of search space, a personalized query can be regarded as a range query whose search space contains its top-k results, with the shape and size as its main factor. The shape of the search range is determined by the personalization specified in the score function, and thus varies from one score function to another. The overall size of the search range (i.e., the region in the domain space which contains no less than the desired k data tuples) can be approximated based on data density estimation in the current system. Intuitively, the personalized query can be resolved by retrieving tuples in the range of a specific shape determined by the personalization in the score function and the data density of the current system. The shape and overall size of the search range become the main cost factors in evaluating the personalized query. This paper focuses on these two aspects. The key idea is to transform a personalized query into a range query on-the-fly with a specific range shape based on the personalization and then retrieve the top-k results from the determined search range based on the multiscale data density estimation in the current P2P system. Our main contributions include:

- We extend our system overly [21] for personalized query evaluation, which is slightly modified from the existing DHT-based P2P overlay networks (e.g., Chord [31] and Pastry [26]). Inheriting the properties of the existing P2P overlay networks, our system overlay further benefits data indexing and accessing by involving gray code based indexing strategy. Properties of our data indexing method are discovered, proved and utilized in the personalized query evaluation.

- We present the first method which evaluates general personalized queries in the P2P literature. To achieve efficient evaluation, we transform a personalized query into a range query with a specific shape based on the personalization given in the score function and the data distribution estimated in the current system. Based on matrix singular value decomposition (SVD), a new method which transforms the shape of the search range for a personalized query with a quadratic score function from an arbitrary hyper-ellipsoid into a simple hyper-sphere is developed. By fully using the estimated multiscale data density information, the range which contains the desired top-k results can be selected efficiently at run time, due to its zoom in and zoom out capability. A multicast routing algorithm is also applied to retrieve all the candidate tuples with minimal communication cost.

- We propose a novel multiscale density estimation algorithm to summarize the data distribution across the current network, which helps to approximate the overall size of the search range for a personalized query. Data density is hierarchically summarized in terms of scale levels, each of which partitions the entire data space into different scales and records the data density in the corresponding scale range. Each peer maintains partial density information at each scale level and contributes to the overall multiscale data density in the network. Data density update and maintenance in the dynamic P2P networks are also discussed with theoretical analysis.

- We conduct an extensive performance study which shows the effectiveness and efficiency of our methods for personalized query evaluation in P2P systems.
2. Related work

Generally speaking, personalized queries are closely related to preference queries (i.e., simplified score functions) and \( k \)-\( NN \) queries (i.e., static score functions), while both of them can be regarded as special cases of the personalized query.

2.1. Preference query

Preference query evaluates a user’s request in terms of his individual preference. Herein, the preference can be both quantitative and qualitative, and consequently categorized into quantitative evaluation (i.e., top-\( k \) evaluation) and qualitative evaluation (i.e., skyline query [4,19]) respectively.

The commonly used score functions for top-\( k \) queries are monotone functions, such as \( f(t) = \sum_{i=1}^{N} w_i + t \cdot a_i \), where higher weight (i.e., \( w_i \)) is given to preferred attribute (i.e., \( a_i \)). Onion [3] is one of the earliest preference query search algorithms. It computes and materializes the convex hull of all layers of the whole data set. Given a query, Onion first scans the most-exterior hull and then progresses to the inner layers incrementally. Since it searches the fewest layers of the convex hull which are necessary, it is optimal. Robust index [39] is another layered index method similar to Onion. It computes the robust index for each data tuple \( t \), i.e., the minimum number of data tuples over which \( t \) ranks according to arbitrary preferences. The two methods mentioned above are based on the index pre-computed from the whole data set. Prefer [13] is one of the most successful systems that deal with linear score functions, materializing a number of views according to different preferences. New preference queries are then evaluated based on the materialized views. If the preferences of a new query are the same or quite similar to the preferences of one materialized view, the new query could be answered quickly. Otherwise, Prefer materializes many views to achieve high performance. Thus data tuples may be indexed several times in different views. Some existing spatial index methods, such as R-tree and its derivations like R*-tree can also be used to facilitate preference query processing [17,33]. Other work also extends the score function to ad hoc functions. Ref. [40] gives a solution for constraint non-monotone score function based on threshold algorithms (i.e., TA) [10,20], which use a sort-merge framework that assumes data lists are pre-sorted. For multi-dimensional data, each tuple is pre-sorted in two lists. The front parts of the two lists are frequently accessed. Ref. [41] computes the top-\( k \) results for ad hoc score function in terms of selection multi-dimensions based on ranked cubes. In this work, each preference dimension is partitioned in equal-depth bins, and data tuples are grouped into grids by intersecting the bins from all participating preference dimensions. During top-\( k \) query execution, the grid which contains the extreme point is first located. Its neighboring grids are then progressively expanded.

Skyline evaluates preference query qualitatively. The recent work SPEERTO [34] employs a layered index method (i.e., K-skyband [24]) in the unstructured P2P system organized with a super-peer network. Each super-peer maintains and aggregates the K-skyband sets of its peers to answer any incoming top-\( k \) query. By exchanging a skyline set at super-peer level, it can provide the exact and complete result set in a progressive way. Skyframe [38] is a framework for efficient skyline query processing in P2P systems which considers response time, communication cost and load balancing issues. However, skyline query processing methods are not directly applicable for the general preference query as defined later in Section 3.

The score functions used in preference queries are special cases of those used in personalized queries to indicate both personalization and current search requirement. Moreover, the solution for preference queries are not extendable for personalized query in dynamic P2P networks.

2.2. \( k \)-NN query

\( k \)-nearest neighbor (\( k \)-\( NN \)) search [18] finds the \( k \) objects in the data set which have the \( k \) smallest distances under a specific distance function (e.g., score function) \( d \) with respect to the query object \( q \). Usually, Euclidean distance and Hamming distance are used as the metric to evaluate the distance between the objects in the data set and the given query object. \( k \)-\( NN \) search in the P2P literature is the most related work to our’s, while the distance function is static selected for one system. Furthermore, the mainly used Euclidean distance (e.g., VBI-Tree [14]) failed to take dependency between attributes (or dimensions) into account. iDistance [15] is an alternative index strategy chosen to find the nearest neighbor (e.g., M-Chord [22] and [8]), whose distance function is consequently degraded to Euclidean distance too.

The personalized query and \( k \)-\( NN \) query mainly differ in: (1) The variability of the score function. The score function is variable on-the-fly for each query in the personalized query system, while it is statically specified for the \( k \)-\( NN \) query system. Furthermore, the mainly used Euclidean distance (e.g., VBI-Tree [14]) failed to take dependency between attributes (or dimensions) into account. iDistance [15] is an alternative index strategy chosen to find the nearest neighbor (e.g., M-Chord [22] and [8]), whose distance function is consequently degraded to Euclidean distance too.

The personalized query and \( k \)-\( NN \) query mainly differ in: (1) The variability of the score function. The score function is variable on-the-fly for each query in the personalized query system, while it is statically specified for the \( k \)-\( NN \) query system. Furthermore, the mainly used Euclidean distance (e.g., VBI-Tree [14]) failed to take dependency between attributes (or dimensions) into account. iDistance [15] is an alternative index strategy chosen to find the nearest neighbor (e.g., M-Chord [22] and [8]), whose distance function is consequently degraded to Euclidean distance too.

In this paper, we evaluate the personalized query with a more generalized score function, indicating both users’ personalization and current search requests.
2.3. Data distribution estimation

The data distribution refers to the frequencies of data occurrence over its entire domain, i.e., each dereferencing within the domain associated with a frequency indicating the number of data tuples on it. The data distribution estimation plays an important role in our system for transforming a personalized query into a range query. Basically, the estimation methods can be divided into four categories. First, the parametric method by model function [27] achieves high estimation accuracy when knowing the distribution model in advance. Second, the curve fitting method by general polynomial functions [2] which is able to approximate any data distribution but for the error propagation problem. Third, the sampling method [44] which achieves unbiased sampling by transforming any distributions to a intermediate uniform distribution (e.g., cumulative distribution). Forth, non-parametric methods such as discrete cosine transform [43] and density kernel [42] are more robust in estimation while suffer from high a computation cost.

In this paper, we proposed the multiscale densities which summarize data distribution in different scale levels, and affiliates the transformation from personalized query to range query at an acceptable cost.

3. Preliminary

In this section, we formally define the research problem to be solved and provide an overview on the P2P system with designed network overlay and data indexing.

3.1. Problem statement

We assume a tuple of data is represented by a sequence of attribute-value pairs: $t(a_i, v_i), i = 1, \ldots, N$. The domain $A_i$ of attribute $a_i$ is either numerical or categorical. A numerical domain is normalized into the range $[0,1]$.

The queries to be evaluated are top-$k$ queries, while the score function is personalization-oriented. Personalization which is indicated by the user or learned from the user’s search history, is integrated into the score function to consider the weights of (cross) attributes. A commonly used score function is based on the quadratic distance function which is more versatile in specifying users’ intentions, since it also considers the dependencies across attributes. Formally, we use distance Formula (1) as the score function.

$$d(t) = (t - q) \cdot M \cdot (t - q)^T$$

where $M$, named personalization matrix, is a square matrix specifying the personalization (or weights) which varies from one user (query) to another, while $t$ is a data tuple to be evaluated, and $q$ is the query tuple specified by the user, indicating a user’s current search intention. Herein, $M$ is a $N \times N$ matrix, with each of its diagonal element indicating the preference on the corresponding single attribute and other elements for the preferences across attributes. As $d(t)$ measures distance, $d(t) \geq 0$. Obviously, matrix $M$ is positive-semidefinite since $d(t) \geq 0$ for all possible $(t - q)$.

Evaluating a personalized query is to find the top-$k$ data tuples that minimize the specified score function with $M$ in the current dynamic system. The main difference between our personalized query and $k$-nearest neighbor ($k$-NN) query with quadratic distance function lies in: the matrix $M$ varies from one user (query) to another in a personalized query, while it is set static in the $k$-NN query. In this paper, we aim to process the personalized query in P2P environments effectively and efficiently. The process of evaluating a personalized query in P2P systems contains two main components: data density estimation and range query transformation, which are introduced in Section 4 and Section 5 respectively.

3.2. System overview

The overlay network of the P2P system used for personalized search is the one proposed in our previous work [21], by slightly modifying the existing DHT-based P2P overlay network (e.g., Chord [31] and Pastry [26]). To make the paper self-contained, we provide a brief overview of the P2P system in this subsection, including its network overlay and data indexing. Herein, we mainly focus on ring-based overlays. It should be noticed that algorithms presented in this paper are applicable for content-addressable networks (i.e., CAN). In terms of the finger table construction (Section 3.2.1) and data index based on domain partition (Section 3.2.2), which are consistent with those in content-addressable networks (i.e., CAN), algorithms for personalized query evaluation presented in this paper could be extended to content-addressable networks (i.e., CAN) in trivial ways.

3.2.1. Network overlay

Our network overlay is ring-based by slightly expanding existing P2P network overlays such as Chord [31] and Pastry [26]. Each peer in the system is identified by a $p$-bit binary sequence called peer id, where $p$ is a positive integer and is decided by the size of the network. The whole permutation set of $p$-bit binary code constructs a large self-contained ring which could accommodate $2^p$ peers, as shown in Fig. 1. Peer with id $i$ contains two finger tables which maintain links to other peers in the ring, i.e., finger table and inverse finger table (ifinger table). Inverse finger table affiliates in fetching the desired top-$k$ results (see Section 5.2). The $j$th finger in the finger table on peer $i$ links to the first peer whose id is equal to or follows
by \(i + 2^{j-1}\), denoted as successor\((i + 2^{j-1})\). The inverse finger table construction is based on finger table. Each finger in the inverse finger table is the inverse link of the finger in the finger table. The algorithm for inverse finger table construction is given in Algorithm 1. Under such a construction, given a peer with id \(i\), all ids which differ in one-bit with \(i\) can be located in one hop. Take the peer with id 100000 in Fig. 1 as an example, fingers in bold (i.e., 100100, 101000, 110000 in finger table and 000000 in ifinger table) differ in one-bit with their hosted peer id 100000. Therefore, those peers are one hop away from the peer with id 100000. We formally give the theorem on links between peers under our network overlay.

#### Algorithm 1. Inverse finger table construction.

1: for the \(j\)th finger \(\text{finger}[j]\) in the finger table of peer \(i\) do
2: \( \text{ifinger}[j].\text{node} = i \)
3: end for

#### Theorem 1. Given a peer with id \(i\), all ids which differ in one-bit with \(i\) can be located in one hop.

**Proof.** See Theorem 1 in [21].

### 3.2.2. Data indexing

Data indexing contains two aspects, i.e., index key generation and index insertion.

An index key for a multi-attribute tuple is a \(p\)-bit identifier which is combined from index codes of a tuple on all its attributes. The number of bits used to represent index code for attribute \(a_i\) is decided in advance, denoted by \(p_i\). Additionally, the formula \(\sum_{i=0}^{N} p_i = p\) holds. After decided the number \(p_i\), we partition the domain of \(a_i\) into \(2^{p_i}\) intervals equally. One \(p_i\)-bit gray code [12] is then attached to each interval sequentially. As an example shown in Fig. 2, the domains of \(a_1\) and \(a_2\) are divided into 8 intervals encoded with gray code sequentially. Under such a partition, the data space is presented by a number of non-overlapping grids, such as grids \(A\) and \(B\) in Fig. 2. The index code used to represent the attribute value is the gray code attached to the index interval in which the attribute value lies. For example, grid \(A\) in Fig. 2 has index codes 011 and 111 on \(a_1\) and \(a_2\) respectively.

We use gray code to identify each interval on the partitioned attribute domain because of its one-bit difference property (see Lemma 1 below), which is consistent with Theorem 1.

#### Lemma 1. Adjacent gray codes in the grad code sequence differ in one bit only.

**Proof.** See Property 1 in [6].
As the one-bit difference in both Theorem 1 and Lemma 1, the combination of them leads to Properties 1–4, and results in eventual but notable reduction in communication cost.

The index key generation for a data tuple is based on all its index codes, denoted as key. We provide a shuffled-based method to generate index keys for data tuples, which enables peers to maintain a set of index keys which contribute to a range with similar ratios of its entire domain range for each attribute (see Theorem 2 in detail). Consequently, balanced peer selectivity (the number of peers to be accessed) in evaluating personalized queries could be achieved (considering the arbitrary shaped query range transformed, see Section 5). The shuffled index key is constructed by concatenating a bit of code of one attribute by that of another. The order of attributes concatenation is pre-determined. For example, the index key for grid A in Fig. 2 is 01111. The index key is constructed by concatenating one bit of index code of attribute \( a_1 \) followed by one bit of index code of attribute \( a_2 \). After the key of a tuple is generated, it is inserted into one peer in the network whose id is equal to or follows by its key (i.e., successor(key)). Using such a data indexing method, we get the following properties.

**Property 1.** Index keys for two adjacent grids in the multi-dimensional data space differ in one bit only.

**Proof.** See Property 2 in [21].

**Property 2.** Grids whose index keys differ in one-bit are linked together by links in (inverse) finger table.

**Proof.** See Property 3 in [21].

Prefix of gray codes for adjacent intervals on one attribute can be extracted, as shown in Fig. 2. For example, prefix “00” is extracted from two adjacent intervals 000 and 001 on attribute \( a_1 \), and other prefixes are extracted as the same. For the prefix, we have following property:

**Property 3.** Extracted prefixes preserve the gray code permutation sequence. Assume the length of the extracted prefixes is \( p_i \) bit (\( p_i \leq p \)), which are extracted from attribute \( a_i \), then each prefix identifies one of the \( 2^p \) equal partitioned intervals on attribute \( a_i \).

**Proof.** See Property 4 in [21].

As given in Property 3, prefixes identify intervals under different sized equal partitions on the attribute domain. Regions which can be identified by concatenation of prefixes on each attribute are called scales, i.e., a set of continuous grids in the data space. For example, the region which is marked with “\( \dagger \)” in Fig. 2 has prefixes 00 and 00 on attribute \( a_1 \) and \( a_2 \) respectively, resulting in its identifier 0000 consequently. Another example is the region which is marked with sparse cross, and identified by 110, has prefixes 1 and 10 on attribute \( a_1 \) and \( a_2 \) respectively. Clearly, the data space is divided into different scales according to different lengths of prefixes on attributes, as shown in Fig. 2. We refer to a scale which is identified by the same length of prefix on all attributes (e.g., “\( \dagger \)” region in Fig. 2) as the equal-scale which has the following property.

**Property 4.** Equal-scales have the same properties as grids, i.e., Property 1–3.

**Proof.** See Property 5 in [21].
The properties listed above are important, since they will be used for data density estimation presented in Section 4 as well as personalized query evaluation presented in Section 5. For simplicity, in the rest of the paper, we refer to equal-scale as scale. Meanwhile, since grid can be regarded as the lowest level scale, we will use scale consistently without ambiguity. For clarity, we list the concepts used frequently in this paper in Table 1:

**Theorem 2.** Shuffle-based gray code concatenation for index key generation enables peers to maintain a set of index keys which contribute to a range with a similar ratio of the entire domain range on each attribute.

**Proof.** As assumed, the network contains \( n \) peers, which are distributed sparsely in the ring (i.e., \( n \ll 2^p \)). Without lose of generality, we assume \( l < \log_2 n < l + 1 \), where \( l \leq p \) and \( l > N \). In terms of Property 3, the first \( l \) bits in the index key identifies one scale in \( 2^l \) which are equally partitioned from the entire data space. We call this as an effective scale. Resulting from the index key insertion method, each peer maintains a set of successive index keys, which consequently contribute to such an effective scale. Suppose \( |A_l| = x \), and \( l \mod N = y \). According to the shuffle-based concatenation method, each \( (x + 1) \)-bits in the first \( l \) bits of index key identifies an equal interval of the first \( y \) attributes individually in terms of the concatenation order, i.e., \( a_1, \ldots, a_y \). Similarly, each \( x \)-bits in the first \( l \) bits of the index key identifies an equal interval of the last \( N - y \) attributes individually, i.e., \( a_{y+1}, \ldots, a_N \). That’s to say each peer maintains a set of index keys which contribute to a range with a similar ratio of the entire domain range on each attribute. \( \square \)

**Theorem 2** enables balanced peer selectivity in personalized query evaluation. As peers in the system maintain index keys which contribute to a range with a similar ratio of its domain on each attribute, similar peer selectivity could be achieved for the transformed arbitrary shaped range query.

The existence of Properties 1–4 and Theorem 2 is the reason why we choose gray code and the shuffled-based method to generate an index key.

### 4. Multiscale density estimation for multi-dimensional data

The personalized query is evaluated by transforming it into a designated range query (i.e., a set of desired scales, see Section 5 for detail), which is calculated based on the multiscale density estimation presented in this section. In this section, we discuss the data density estimation in the dynamic network. Compared to our previous work [21], the estimation algorithm proposed in this section is tailored for personalized query evaluation, with much lower storage, communication and computation cost.

**Multiscale densities** maintain the statistical information for scales in the data space. It counts the number of data tuples filled in each corresponding scale in different levels of the multi-dimensional space. By maintaining multiscale densities in each peer, each peer is able to estimate the multi-dimensional data distribution directly and evaluate the personalized query correspondingly. Given the ring-based network overlay introduced in Section 3.2, each peer can maintain multiscale densities for a small portion of data space only. The construction of the multiscale densities is given in subSection 4.1, while its update and maintenance are discussed in subSections 4.2 and 4.3 respectively.

#### 4.1. Multiscale density construction

As provided in Section 3.2.2, the number of grids in the entire multi-dimensional space is \( 2^p \), where \( p \) is usually set to be 32, 64, 128 and so forth. It is infeasible to maintain all possible \( 2^p \) grids on each peer. For the purpose of tradeoff, we maintain densities for scales in different levels (multiscale density in short) on each peer. Next, we introduce how to construct multiple scales in detail, which we have already used in Sections 3 and 5.

The grid introduced in Section 3.2.2 is the finest partition to the entire \( N \)-dimensional space, and we refer to it as level-0 scale. To obtain level-1 scale, we merge two grids which are successive on one attribute (e.g., \( a_i \)). The common prefix of the two grid ids is extracted to be the id of the corresponding level-1 scale. Similarly and recursively, we merge every two scales in level \( l \) which are successive on one attribute to get a coarser scale in level \( l + 1 \). Therefore, each scale in the same scale level is equal in size. The scale density is defined as the number of data tuples in the scale.

**Example 2.** Recall the example in Fig. 2. As grid 000000 and grid 000001 are successive on \( a_1 \), merge the two grids together to the new scale in level 1 with id 0000, whose range is marked “/” in Fig. 2. Obviously, the id for the new merged scale is

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Basic concepts and their definitions.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concept</td>
<td>Definition</td>
</tr>
<tr>
<td>Grid</td>
<td>A unit in ( 2^p ) non-overlapping units after data space partition</td>
</tr>
<tr>
<td>Scale</td>
<td>A coarser region containing a set of continuous grids with the same id prefix</td>
</tr>
<tr>
<td>Equal-scale</td>
<td>Scales having the same length of prefix</td>
</tr>
<tr>
<td>Level-1 scale</td>
<td>A scale after ( l ) time merging from the grids, containing ( 2^l ) grids</td>
</tr>
</tbody>
</table>
the common prefix of its two sub-grid ids (say the last bit in id is discarded). Similarly, the 64 grids in scale level 0 are
merged into 32 scales in level 1 on \(a_1\). Recursively, we are able to merge scales in level 1 into scales in level 2. Scale 00010
and scale 00011 are successive on \(a_2\), which are merged together to be a scale in level 2 with id 0001 (i.e., the range marked
"\(\)" in Fig. 2). Again, the id of the new merged scale is the common prefix of its two sub-scale ids one level lower. The scales
marked with integers (say 0, 1, 2, \ldots, 15) are 16 scales in level 2.

To learn the data distribution in the current network for the sake of personalized query evaluation, each peer needs to
maintain and keep track of the data distribution using scales. To further release the burden of both high storage cost and
communication cost, each peer maintains density information for scales at different levels. This also increases the flexibility
in evaluating different sized range queries (transformed from personalized queries). Next, we focus on the density computa-
tion over different scale levels.

To compute the scale density at a higher level, which contains the information for sum of densities of two sub scales of one
level lower, the following formula is applied:

\[ S_j(l + 1) = S_{2j+1}(l) + S_{2j}(l), \quad j = 0, \ldots, 2^{p-1} - 1 \]  

(2)

where \( j \) is the id for the new merged scale in level \( l + 1 \). For example, \( S_0(1) = S_0(0) + S_1(0) \), \( S_1(1) = S_2(0) + S_3(0) \), and so forth. An
important point to notice is that each \( S_j(l + 1) \) contains density information originating from both \( S_{2j}(l) \) and \( S_{2j+1}(l) \). In other
words, it includes densities from two adjacent scales one level lower. Obviously, a higher level scale density is a coarser esti-
mation based on its lower level scale densities.

In order to recover the lower level densities, we provide another information set called difference of densities, as defined
below:

\[ D_j(l + 1) = S_{2j+1}(l) - S_{2j}(l), \quad j = 0, \ldots, 2^{p-1} - 1 \]  

(3)

where \( j \) is the id for the new merged scale in level \( l + 1 \). For example, \( D_0(1) = S_1(0) - S_0(0) \), \( D_1(1) = S_3(0) - S_2(0) \), and so on. Formula (3) extracts the difference of densities between two scales at a lower level.

Based on Formula (2) and (3), we get two information sets, one for the summed densities and the other for the difference of
densities between two successive scales. Hence, the densities for the two lower level successive scales can be recovered by:

\[ S_{2j+1}(l) = [S_j(l + 1) + D_j(l + 1)]/2 \]  

(4)

\[ S_{2j}(l) = [S_j(l + 1) - D_j(l + 1)]/2 \]  

(5)

As \( S_{2j+1}(l) \) and \( S_{2j}(l) \) can be recovered, we could discard them and maintain \( S_j(l + 1) \) and \( D_j(l + 1) \) only instead. Recursively, we
apply the Formulas (2) and (3) to the sum of densities (i.e., \( S_j(l) \)) of scales from the lowest level (i.e., level 0) to the highest
level (i.e., level \( p \)). The entire information should be maintained to recover density information for scale in all levels are as
following set: \( \{ S_0(p), D_0(p), D_1(p - 1), D_2(p - 1), \ldots, D_{2^p - 1}(1), \ldots, D_0(1) \} \).

For density information recovery, we apply Formulas (4) and 5. For example, \( S_1(p - 1) = [S_p(p) + D_p(l + 1)]/2 \) and
\( S_0(p - 1) = [S_p(p) - D_p(l + 1)]/2 \). Recursively, we could get density information in all levels. Note that no information is lost
relying on the above transformation.

Hence, with the same storage cost to maintain each grid density, we are able to keep track of the density information for
every scale in all levels. Furthermore, this multiscale density computation algorithm results in Theorems 5 and 6, which affili-
late the personalized query evaluation in dynamic P2P networks.

**Example 3.** Take Fig. 2 again as an example. Suppose the data tuple to be managed in our system contains two attributes.
Considering their domains, we partition each domain into 8 equal intervals which are encoded as gray codes sequentially.
The entire two dimensional data space is covered with 64 non-overlapping grids (i.e., 64 scales in level 0), as shown in Fig. 2.
According to the scale merge, we get 16 scales in level 2, each of which contains four grids and is marked by an integer (e.g.,
0, 1, 2, \ldots, 15). As encoded with gray codes, the scales in level 2 are locality preserved when they are indexed in the ring-based
overlay network (or content-addressable network, i.e., CAN). To compute multiscale densities, we use the sample data shown
in Fig. 3, where the 16 nodes marked from 0 to 15 are level-2 scales corresponding to those in Fig. 2. Fig. 3 shows the four
levels multiscale density computation (as space limitation, we start from level 2). For scale level 3, we apply Formula (2) to
get to the summed density information which is listed below the line arrows, and apply Formula (3) to get the density
difference between two successive scales, which is listed below the dashed line arrows. When applying Formulas (2) and (3)
to the new computed summed density information recursively, we get \( S_j(l) \) and \( D_j(l) \) for level 4 to 6. Density for the scale in
level 3 is the sum of densities for two successive scales in level 2. For example, the density for the first scale in level 3 (i.e.,
\( S_3(3) \)) in Fig. 3 is the sum of densities for scales \( S_0(2) \) and \( S_1(2) \). For density differences, they are computed similarly. For
information maintenance, the system only maintains the set of \( \{ S_6(6), D_6(6), D_5(5), D_4(5), \ldots, D_2(1), \ldots, D_0(1) \} \) based
on which density information in all levels could be recovered. Considering density recovery for scales in all levels, it goes as
follows recursively. For example, density for scales in level 5 (i.e., \( S_5(5), S_1(5) \)) are recovered by applying the Formulas (4) and
5, say \( S_5(5) = [S_6(6) - D_6(5)]/2 \), \( S_1(5) = [S_6(6) + D_6(5)]/2 \). Similarly and recursively, \( S_4(4), \ldots, S_2(4), \ldots, S_0(0), \ldots, S_0(0) \) could
be recovered.

According to the multiscale density construction algorithm described in this subsection, we provide the following Theo-
rem 3 to give a brief summary.
Theorem 3. Together $2^p$ entries (i.e., \{$S_0(p), D_0(p), D_1(p - 1), D_0(p - 1), \ldots, D_{2^{p-1}-1}(1), \ldots, D_0(1)\}$) are maintained to preserve the density information for all scales at all levels with no information loss.

Proof. We firstly prove that information maintained for density preservation needs $2^p$ entries.

As the total number of scales in level 0 over the entire data space is $2^p$, we need to maintain $2^p$ densities (i.e., one for each scale). For scale level 1, we apply Formulas (2) and (3), and get $2^{p-1}$ merged sum of densities (i.e., $S_0(1), \ldots, S_{2^{p-1}-1}(1)$) and $2^{p-1}$ difference of densities (i.e., $D_0(1), \ldots, D_{2^{p-1}-1}(1)$). For information maintenance, we discard the $2^p$ densities in level 0 and maintain the $2^{p-1}$ sum of densities and $2^{p-1}$ difference of densities in level 1 instead (total number is $2^p$ too). Assume we need to maintain $2^{p-1}$ merged sum of densities and $\Sigma_{i=0}^{2^{p-1}-2}2^{p-i}$ difference of densities in level l. Then for level $l + 1$, we simply apply the Formulas (2) and (3) to information maintained for level l, and get $2^{p-2}/2 = 2^{p-l-1}$ merged sum of densities and $\Sigma_{i=0}^{2^{p-l-1}-1}2^{p-i}$. Hence, the total number maintained for density information preservation is $\Sigma_{i=0}^{2^{p-1}-1}2^{p-i} + 2^{p-l-1} = 2^p$.

Secondly, we prove the information maintained has no information loss for all scales in all levels. The density information for a scale in level $p$ (i.e., $S_p(p)$) is maintained already. The densities information for level $p - 1$ are recovered from $S_0(p)$, $D_0(p)$. Assume density information for scale in level l (i.e., $S_l(l), \ldots, S_{2^{p-l}-1}(l)$) could be recovered. Then for level $l + 1$, densities for scales $S_0(l-1), \ldots, S_{2^{l+1}-1}(l-1)$ from $S_0(l), \ldots, S_{2^{l+1}-1}(l)$ and $D_0(l), \ldots, D_{2^{l+1}-1}(l)$ which are maintained. □

Armed with Theorem 3, we could estimate data density precisely with no storage cost overhead while obtaining the flexibility for personalized query evaluation.

4.2. Multiscale density update

In this subsection, we introduce how to update the multiscale density.

The P2P network is a high dynamic system which involves a high change rate for both node status (i.e., join, leave and fail) and data tuple status (i.e., insertion and deletion). As the change of node status causes both data movement (e.g., join invokes a specific data tuple to be moved to the new joined node) and data update in the corresponding scales in level 0 (e.g., join brings a set of data tuple to be inserted to the specific scales), we separate it into two parts, one for data maintenance (i.e., data movement) which will be given in the next subsection and one for data update (i.e., data change). We will focus on data update in this subsection.

For data update, we maintain a multiscale density tree which is the same as given in Fig. 3. Repeatedly applying Formulas (2) and (3), we are able to get the multiscale density for the updated data in the corresponding scale levels, which are denoted as $\Delta S_j(l)$ and $\Delta D_j(l)$ similarly. To get the new updated density information, we have the following theorem.

Theorem 4. (Linear Theorem) Denote the density information for scale $j$ in level $l$ before update as $S_j(l)$ and $D_j(l)$, and the density information to be updated as $\Delta S_j(l)$ and $\Delta D_j(l)$. The density information (i.e., $S_j'(l), D_j'(l)$) after update are:

\[ S_j'(l) = S_j(l) + \Delta S_j(l) \]
\[ D_j'(l) = D_j(l) + \Delta D_j(l) \]
Proof. Denote the densities for scales $S_{2j}(0)$ and $S_{2j+1}(0)$ before update as $S_{2j}(0)$ and $S_{2j+1}(0)$, and the densities being updated as $ΔS_{2j}(0)$ and $ΔS_{2j+1}(0)$ respectively. Then, we get:

$$S_j'(1) = [S_{2j-1}(0) + ΔS_{2j-1}(0)] + [S_{2j}(0) + ΔS_{2j}(0)] = [S_{2j-1}(0) + S_{2j}(0)] + [ΔS_{2j}(0) + ΔS_{2j}(0)] = S_j'(1) + ΔS_j'(1)$$  

(8)

with

$$S_j'(l) = S_j(l) + ΔS_j'(l)$$

we get

$$S_j'(l + 1) = [S_{2j-1}(l) + ΔS_{2j-1}(l)] + [S_{2j}(l) + ΔS_{2j}(l)] = [S_{2j-1}(l) + S_{2j}(l)] + [ΔS_{2j}(l) + ΔS_{2j}(l)] = S_l(l + 1) + ΔS_l'(l + 1)$$

(9)

The proof to the $D_j'(l)$ is similar. □

Theorem 4 gives the instruction for data update, which is updated linearly in the system with a low computation cost.

4.3. Multiscale density maintenance in P2P systems

In subSections 4.1 and 4.2, we give an overview to the theorem of multiscale density estimation over the entire system, including both construction and update. However, in P2P dynamic systems, each peer maintains a small set of index, and consequently manages a small set of the multiscale density information, which contributes to constructing the entire multiscale density over all scales in all levels. In this subsection, we drill down to investigate the multiscale density maintenance on each peer. Several properties are discovered, which lead to low storage cost and communication cost in density estimation.

Two scenarios of maintenance are introduced, one for density information construction and the other for data movement due to the status of changing of peers or demand on load balancing (peer movement in the network [45]).

For density information construction, two sets of data are used, one for the density information before data update and the other for data being updated. Each peer in the network maintains a multiscale density table, each entry of which contains the density information (i.e., the differences of densities at all levels and the top level density). The multiscale density table in Fig. 1 gives an intuition.

Entries in the multiscale density table are computed hierarchically. Algorithm 2 provides the details. Let’s take the multiscale density computation in peer with id “001100” in Fig. 3 as an example to understand Algorithm 2 clearly. Suppose peer with id “001100” maintains three scales in level 2 locally (for space limitation, we start from level 2), say $S_{22}(0)$, $S_{22}(1)$, and $S_{22}(2)$. All the computations fall into three scenarios. Computing $S_l(3)$ needs to know $S_{42}(2)$ and $S_{42}(2)$ which are maintained on its previous peer. Line 10 and 11 in Algorithm 2 give instruction on computing $S_l(3)$. Line 5 shows how to compute $S_l(3)$ since $S_{42}(2)$ and $S_{42}(2)$ are maintained locally. $S_l(3)$, $S_{42}(3)$ and corresponding $D_l(3)$, $D_{42}(3)$ contribute to the density information of scales in level 3, and we maintain $D_l(3)$, $D_{42}(3)$. Similarly, we could compute $S_l(4)$ and $S_l(4)$ based on line 7 to 8 in Algorithm 2. From Algorithm 2, we can compute all the entries in the multiscale density table. In the end, peer “001100” maintains the values which are marked with gray rectangles and below the dash lines in Fig. 3. Density information for scales in all levels for peer “001100” can be recovered based on Formulas (4) and (5). Theorem 5 gives the instruction on the entries which should be maintained in the multiscale density table for peer i.

Theorem 5. Assume peer $i'$ in the network has peer i as its successor, then peer i maintains the multiscale density information as follows: \{D_{ij}(1),…,D_{ij}(1)} on level 1, \{D_{ij+1}(2),…,D_{ij+1}(2)} on level 2, …, \{D_{ij}(p - 1), {D_{ij}(p)}, S_{ij}(p)} on level p. That’s to say peer i maintains difference of density (i.e., $D_i(l)$) for all scales in all levels to which it contributes its index.

Proof. Peer i maintains index keys from $i' + 1$ to i. For multiscale density construction on level 1, it computes $S_{ij}(1),…,S_{ij}(1)$ and $D_{ij}(1),…,D_{ij}(1)$ in terms of Algorithm 2. Peer i stores $D_{ij}(1),…,D_{ij}(1)$, and uses $S_{ij}(1),…,S_{ij}(1)$ for further computation on level 2. Consequently, it gets $S_{ij}(2),…,S_{ij}(2)$, and $D_{ij}(2),…,D_{ij}(2)$ for level 2. Simply and recursively, peer i stores \{D_{ij}(1),…,D_{ij}(1)}, \{D_{ij}(2),…,D_{ij}(2)},…,\{D_{ij}(p - 1), {D_{ij}(p)}, S_{ij}(p)} □

Example 4. Take peer 001100 in Fig. 3 as an example. It maintains scales 3,4,5 in level 2 (as space limitation, we start from level 2). Then, it computes and stores $D_{32}(3),D_{32}(3)$ in level 3 since it contributes indexes to Scale 001 and Scale 010. Similarly, it computes and stores $D_{03}(4),D_{03}(4)$ in level 4 because of its index contribution to Scale 00 and Scale 01.

As we known, peers in the network are very sparse and each of them maintains a set of successive index keys. Thus, density of scales in lower levels could be computed locally (say without information exchange with other peers in the network).
Theorem 7. When there is overlapping, what about the average storage cost for density information maintenance of each peer in the network? For example, density information for Scale 01 in level 4 will be maintained by both peer 001100 and its successor. As such, the density information (i.e., difference of density) for this scale will be maintained on more than one peer.

Scale 001 in level 3 partially. For any scale which is partially contributed by one peer, it must have other contributor(s). For the rest of this theorem, we start from level 2 contribute to Scale 010 in level 2 completely. Similarly, Scale 0011 in level 2 contributes to Scale 0010 in level 2.

Algorithm 2. Multiscale density computation.

1: for entry[i] in the multiscale density table of peer i do
2: //entry[i] means the i-th entry in the multiscale density table, including \( (D_j(l))_{j=0}^{2^p-1} \), \( l \in [1, p] \).
3: for Each j do
4: if \( S_{2j+1}(l-1) \) & & \( S_{2j}(l-1) \) are on peer i then
5: Get local \( S_{2j+1}(l-1) \), \( S_{2j}(l-1) \)
6: else if \( S_{2j}(l-1) \) is on peer i then
7: Send \( S_{2j}(l-1) \) to finger[i].node
8: \( S_{2j+1}(l-1) \) received from ifinger[i].node
9: else
10: Send \( S_{2j+1}(l-1) \) to ifinger[i].node
11: \( S_{2j}(l-1) \) received from finger[i].node
12: end if
13: store \( D_j(l) = S_{2j+1}(l-1) - S_{2j}(l-1) \)
14: end for
15: end for

Given that each peer in the network maintains hierarchical density information from level 1 to level \( p \), different scale levels can be examined depending on the personalized query and density information. By maintaining a small set of differences of densities at all levels, the exact density information at each level can also be recovered. Take a peer with id 001100 in Fig. 3 as the example again. It should have density information for \( S_6(2), S_6(2), S_6(2) \) in scale level 2, \( S_6(3), S_6(3) \) in scale level 3, \( S_6(4), S_6(4) \) in scale level 4, \( S_6(5) \) in level 5 and \( S_6(6) \) in level 6 respectively. All the information could be recovered from the information maintained in \( S_6(6), D_6(6), D_6(5), D_6(4), D_6(4), D_6(3), D_6(3) \). See Theorem 6 as below:

Theorem 6. Peer is able to recover the density information for any scale in any level to which it contributes. Its index is based on multiscale information stored locally.

Proof. It could be proved in the reverse way for Theorem 5. We omit for brevity. □

Theorem 6 is very important in personalized query evaluation which indicates no communication overhead in computing detailed local scale density.

On the other hand, the set of successive non-overlap index keys maintained on one peer contribute to a set of scales in different levels completely or partially. Take peer 001100 in Fig. 3 as an example. Scale 0100, 0101 in level 2 (as space limitation, we start from level 2) contribute to Scale 010 in level 2 completely. Similarly, Scale 0011 in level 2 contributes to Scale 001 in level 3 partially. For any scale which is partially contributed by one peer, it must have other contributor(s). Consequently, the density information (i.e., difference of density) for this scale will be maintained on more than one peer. For example, density information for Scale 01 in level 4 will be maintained by both peer 001100 and its successor. As such overlapping exists, what about the average storage cost for density information maintenance of each peer in the network? Theorem 7 gives this in detail.

Theorem 7. The average storage cost for each peer on multiscale density information in the network is within

\[
\left[ \frac{p}{n} + p - \log_2 \frac{n}{p}, \frac{p}{n} + \log_2 \frac{n}{p} \right]
\]

where \( n \) is the number of peers in the network, and \( p \) is the bit-length of peer id.

Proof. Without loss of generality, we assume the successive index key range maintained by peer \( i \) is \( r_i \). According to Theorem 5, peer \( i \) maintains \( r_i + (p - \lfloor \log_2 r_i \rfloor) \) entries for scale density information. Similarly, the entire number of entries maintained across all the peers in the network should be \( \Sigma_{i=1}^n r_i + (p - \lfloor \log_2 r_i \rfloor) \), where \( \Sigma_{i=1}^n r_i = 2^p \). Thus, \( \Sigma_{i=1}^n r_i + (p - \lfloor \log_2 r_i \rfloor) = 2^p + np - \Sigma_{i=1}^n \log_2 r_i \). It gets maximum when \( r_j = 2^p \) and \( r_i = 1 \), \( i \neq j \), \( i \in [1, n] \). So, \( 2^p + np - \Sigma_{i=1}^n \log_2 r_i \leq 2^p + np - \log_2 2^p = 2^p + p(n - 1) \). It gets minimum when \( r_i = r_{i+1} \), \( i \in [1, n] \). So, \( 2^p + np - \Sigma_{i=1}^n \log_2 r_i \geq 2^p + np - \log_2 (\frac{n}{1})^n = 2^p + np - n \log_2 \frac{n}{1} \). □

In terms of Theorem 7, the average storage cost for multiscale density information maintenance is rather low, if suitable \( p \) is selected.

Density maintenance due to data movement is simple. Data are moved to nearby peers in the scenarios of peer status changing (say join, leave, fail) or load balancing. The density information will move to the peer for those scales in levels which it has not maintained before. We omit this for brevity.

Thus, the multiscale density information is constructed and maintained across all peers in the dynamic P2P systems with low storage cost and communication cost as well as scale density recovery.
5. Personalized query evaluation

As discussed in Section 3.2, the entire data space is divided into a set of non-overlapping grids, and the index key of each data tuple falls into one of them. Hence, the number of data tuples in each grid maintains the statistic information about the data distribution in the current system. Grids are further organized into multi-level scales. In this section, we discuss how to process personalized query efficiently based on the estimated multiscale data density (discussed in Section 4). At a scale level, each scale calculates the number of index keys located in the corresponding scale. Therefore, based on those dedicated scales, the search range, i.e., a set of adjacent scales (maybe in multiple levels), which contains the desired top-k results for a personalized query with an arbitrary quadratic scoring function is able to be computed on-the-fly. We call this procedure range computation. After we compute the search range, the next step is to route the request to all the peers maintaining the search range for the desired data tuples, which is called range routing.

5.1. Range computation

The score function of a personalized query is presented in a quadratic form, as shown in Formula (1). We evaluate this general form of score functions since it covers both user personalization (i.e., M) and current query intensions (i.e., q), while it removes an independency assumption between attributes. In a geometric view of such a score function, it is a hyper-ellipsoid in the N-dimensional space. The hyper-ellipsoid hereby is the N-dimensional generalization of an ellipsoid with arbitrary orientation (i.e., the axes of the hyper-ellipsoid are arbitrarily oriented in the space). Obviously, the cost is rather high in figuring out the search range for a query, i.e., the scales which are overlapped with an arbitrarily oriented hyper-ellipsoid.

Algorithm 3. Search range computation.

```java
RangeCompute() {
    initialize a set P to store scales to be searched
    l = ZoomOut(1, k) // get suitable level l starting from 1
    convert the query hyper-ellipsoid into the hyper-sphere v in terms of l
    initialize the enlarging step D
    while results < k do
        for j from min to max do
            // minimum id and maximum id for scale in level j maintained on peer p
            if S_j(l).isAdded = true then continue
            if IsFarthestPointCovered(l, j) then
                P.add(S_j(l))
                S_j(l).isAdded = true
                results += S_j(l)
            else if IsNearestPointCovered(l, j) then
                ZoomIn(results, l, j)
            end if
        end for
        v = v + Δv
    end while
    return P
}

IsFarthestPointCovered(int l, j) {
    if ||S_j(l).farpoint × E_M.A^l||_2 ≥ ||v||_2 then return True
    else return False
}

IsNearestPointCovered(int l, j) {
    if ||S_j(l).nearpoint × E_M.A^l||_2 ≥ ||v||_2 then return True
    else return False
}
```
Fortunately, we find that a hyper-ellipsoid can be transformed into a hyper-sphere based on singular value decomposition (SVD), which significantly relieves the computational cost in figuring out the search range. For the purposes of simple illustration and easy understanding, Fig. 4 gives an intuition on the conversion from an ellipse to a circle in a 2-dimensional space.

As given in Section 3.1, $d(t) \geq 0$ holds, i.e., matrix $M$ is positive-semidefinite. The spectral decomposition of $M$ can be calculated as:

$$M = E_M \Lambda E_M^T$$

(10)

where $E_M$ is the set of eigenvectors of $M$, and the diagonal matrix $\Lambda$ consists of eigenvalues $\{\lambda_M, \lambda_M, \ldots, \lambda_M\}$ of $M$.

With Formula (10), the score function (Formula (1)) becomes:

$$d(t) = (t - q)^T M (t - q) = (t - q)^T E_M \Lambda E_M^T (t - q)^T$$

(11)

Associating components of Formula (11), we get:

$$d(t) = (t - q)^T E_M \Lambda E_M^T (t - q) = [(t - q)^T E_M \Lambda] [(t - q)^T E_M] = \mu^T \mu$$

(12)

where $\mu$ is the eigenvector of diagonal matrix $\Lambda$. It indicates a hyper-ellipsoid. A 2-dimensional example (i.e., ellipse) is shown in Fig. 4a.

Factorizing Formula (11), we get the following formula too:

$$d(t) = (t - q)^T E_M \Lambda E_M^T (t - q) = (t - q)^T E_M \Lambda I^n [(t - q)^T E_M \Lambda]^T \mu v^T$$

(13)

where $v$ is the eigenvector of diagonal identity matrix $I$. It indicates a hyper-sphere.

![Fig. 4. Conversion from ellipse to circle.](image-url)
By Formula (11)-(13), we have:

$$14$$

$$\langle t-q \rangle E_t A_t^2 = \mu A_t^2 = v l$$

where $|v|$ is the radius of the hyper-sphere in the converted data space. By applying Formula (14), a hyper-ellipsoid can be converted into a hyper-sphere. A 2-dimensional example for the transformation from an ellipse to a circle is shown in Fig. 4a and b.

Consequently, calculating the search range which contains the top-$k$ results for a personalized query can be simplified in the converted data space. As we discussed in Section 3.2.2, the data space has been organized into multi-level scales identified by their keys (or ids). The position of a scale in the multi-dimensional space is indicated by its key. Given a personalized query with a parameter matrix $M$ and a query intention $q$ (see Formula (1)), $M$ and $q$ are routed to the peer $p$ which maintains $q$’s index key since peer $p$ knows the densities for scales in different levels around $q$. The query is evaluated there accordingly. Now, the personalized query evaluation is transformed to compute a set of scales which overlap with the hyper-sphere in a proper radius in the converted data space for the dedicated $k$ results.

Based on the knowledge of the space conversion given above, the procedure of computing the search range which contains the top-$k$ results goes briefly as follows: on peer $p$ (which maintains $q$’s index key) iteratively enlarges the radius of the hyper-sphere on a suitable scale level $l$, i.e., $|v|$ in Formula (14). At each iteration, sum up all the densities for the scales overlapping with the hyper-sphere, which are maintained on peer $p$, and record their corresponding scale ids, which will be further used for range routing in Section 4. The iteration stops when the number of results exceeds $k$. Algorithm 3 presents this procedure in detail. In function RangeCompute (), line 1 initializes a set $P$ to store the ids of scales which overlap with the hyper-sphere. Line 2 selects the suitable scale level $l$ to start with the range computation. The resulting level $l$ is given by invoking function ZoomOut $(1, l)$ which recursively computes the sum of densities for scales in a single level maintained on peer $p$ until it reaches $k$. Note that the density for the scale with id $= j$ in level $l$ is denoted as $S_j (l)$. Lines 3–4 convert the query hyper-ellipsoid into the hyper-sphere with radius $v$ in terms of the pre-computed scales in level $l$ and initialize a proper enlarging step $\Delta v$. Lines 5–20 test whether each scale in level $l$ maintained on peer $p$ is entirely or partially overlapped with the hyper-sphere by measuring its farthest (nearest) point. Here, the farthest (nearest) point means one of the end points of scale (scale is basically a hyper-rectangle in the multi-dimensional space) which is farthest (nearest) from the center of the hyper-sphere. Fig. 4c shows four end points for a scale in a 2-dimensional space. If function IsFarthestPointCovered $(l_j)$ is true, i.e., the scale is entirely covered by the hyper-sphere, then it is added to the result set. Else if function IsNearestPointCovered $(l_j)$ is true, i.e., the scale is partially covered by the hyper-sphere, then function ZoomIn () is invoked to examine scales one level lower recursively. Fig. 4c shows examples for entirely covered (i.e., the scale marked with “/”) and partially covered scales by the circle.

Example 5. For a clear understanding, Fig. 4 gives an example for the computation of the scales which are overlapping with the dedicated search range. Suppose we have the data space (i.e., the square) and an initialized search range (i.e., the ellipse) in Fig. 4a. After transformation, the data space is converted into a parallelogram and a circle with radius $v$ for the elliptic search range in Fig. 4b. By testing the farthest point and the nearest point of the converted scale (i.e., in a parallelogram form), we determine whether this scale is overlapping with the search range and calculate the total number of tuples within those overlapping scales. If the total number exceeds $k$, we reduce the radius of the converted circle by $\Delta v$, and calculate the total number again. Otherwise, we reduce the radius by $\Delta v$.

Algorithm 3 returns a set of scales at different scale levels, which contain the desired top-$k$ results. How to fetch all these scales in P2P networks is presented next.

5.2. Range routing

Given a set of scales which are computed previously, in this subsection, we discuss how to route and fetch data tuples in those scales, since the routing is rather important [36].

![Fig. 5. Multicast in the search range.](image-url)
Notice that all the obtained scales contribute to the search range for the query. Based on the index strategy provided in Section 3.2.2, any two adjacent scales at the same level have their keys (ids) differ in one bit (see Properties 1 and 4). The one-bit difference is consistent with that of fingers in the finger table (see Properties 2 and 4). That is to say, adjacent scales could be accessed within one hop in terms of the index maintenance strategy (see Section 3.2.2). In the partitioned space, each inner scale has 2N other scales as its neighbors, i.e., it has two neighbors on each attribute: one is called the positive neighbor (on the top or right) and the other is called the negative neighbor (on the left or bottom). As shown in Fig. 5, C is the positive neighbor of B on \( a_1 \) (the horizontal direction), and A is the negative neighbor of B on \( a_1 \). Next, we provide the multicast route algorithm in the following main steps:

1. Peer \( p \) (which maintains index key \( q \)) sends the request to all its neighbors from the center of the hyper-sphere (i.e., the specific point \( q \) in the query). Its neighbor scales may be maintained locally or by its linked peer in the finger table (Property 2).
2. If a scale receives the request from the positive (negative) neighbor on \( a_i \), then it forwards the request to both its positive and negative neighbors on \( a_1, a_2, \ldots, a_{i-1} \), and forwards the request to its positive (negative) neighbor on \( a_i \).
3. If a scale receives the request which is on the border of the hyper-sphere, then it stops forwarding the request.
4. Each scale caches a small set of routing sequence for the request to prevent it from being forwarded to the scales which have been requested already.

The first three steps in the above algorithm ensure the request to be routed to all the scales which are covered by the desired hyper-sphere and avoid loops. While the P2P system is highly dynamic in nature, the fourth step in the algorithm ensures the request to be routed to the scales once only.

**Example 6.** To have a clearer understanding, Fig. 5 shows an example of the routing sequence for a query circle in a 2-dimensional space, where the lower level scales bounded with dash lines are considered. Firstly, the request is initialized in the center scale X, which is transmitted to its four neighbors (both positive and negative) on \( a_1, a_2 \). The positive neighbor Y of X on \( a_2 \) receives the request and forwards the request to both its positive and negative neighbors on \( a_1 \), and to its positive neighbor on \( a_2 \). The positive neighbor Z of X on \( a_1 \) forwards the request to its positive neighbor on \( a_1 \). Similarly, the request is routed to all the scales and ceases on the border of the circle. With such a multicast routing algorithm, the request could be routed to all the desired scales, and fetch those inside data tuples.

In short, by having the index continuity on the gray code representation of the scale and the ring-based network overlay, personalized query can be efficiently evaluated by quickly allocating those desired scales within the local peer or other peers linked from its finger tables. As for the content-addressable network (i.e., CAN), it has similar properties as those in our modified ring-based network. Hence, the personalized query could be evaluated in a similar way, and we omit this for brevity.

### 6. Performance evaluation

To evaluate the performance of personalized query in P2P networks, we implement one simulator in Java SDK 1.42. In our implementation, each peer is identified by its peer id and maintains two limited message queues, one for sending and one for receiving, which are the same as in Chord [31]. The network layer is simulated to control the network communication, which is the message sending from one peer to another based on peer ids. In our experiments, up to 10,000 peers with randomly distributed peer ids are involved to construct the network ring. The peer id is a 32-bit binary code. On average, about 200 data tuples will be brought into (deleted form) the network when one peer joins (leaves or fails). The simulation time lasts five minutes. Therefore, for a network of 10,000 peers, the average data update rate is approximately 6667 tuples per second. The time for peers to join (leave or fail) is randomly generated. During the life time of a peer in the network, its tuples may be added into (deleted from) it, resulting in the change of the index. The data tuples maintained by peers are multi-attributed, and the attribute types are all numerical, which are generated synthetically. By default, data along each attribute are normally distributed. We also test the performance affected by different data distributions (i.e., normal, uniform, exponential, zipf). In each personalized search, users ask for the top 50 results according to their specific personalization. One hundred personalized queries are tested and their average results are reported.

In P2P literature, the communication cost, the search precision and the search latency are the main considerations. We test these metrics in the dynamic P2P environment and compare with a baseline method by extending the multi-step search which is used in centralized databases.

**6.1. Evaluation in dynamic P2P networks**

In this subsection, we evaluate the performance of our method in dynamic P2P networks. Two dynamic scenarios are designed and tested.
6.1.1. Dynamic Tuples

In this set of experiments, we evaluate the performances of our method in the scenario that data tuples are randomly inserted into or deleted from existing peers. Peers which construct the network of different sizes retain their joined status within the whole evaluation duration. The time for each tuple operation is randomly generated from evaluation start time to end time.

The dispersion of users' personalization is described by coefficient of variation [11] (CV in short) in this set of experiments. In probability theory and statistics, the coefficient of variation is a normalized measure for dispersion of a probability distribution. It is defined as the ratio of the standard deviation \(\sigma\) to the mean \(\mu\), say \(\frac{\sigma}{\mu}\). Hereby, coefficient of variation is used to describe the dispersion of eigenvalues of the personalization matrix \(M\), which is dedicated to the shape of the search range. Therefore, the larger the coefficient of variation, the flatter the hyper-ellipsoid. The flatter the hyper-ellipsoid, there are a great number of scales overlapping with the border of dedicated search range. However, some of the tuples in those scales (e.g., outside of the border of search range) may be ranked lower than the top-\(k\) results, and should be removed after ranking. Hence, the search precision which is defined as the ratio of the top-\(k\) tuples over all fetched tuples is an important parameter in evaluating our system.

The multiscale densities used for maintaining the compact statistical information of data distribution are updated correspondingly, as data tuples are inserted into or deleted from the network frequently. Fig. 6a shows the number of messages needed for the multiscale density maintenance in different sizes of networks (i.e., the number of nodes) and different dimensionalities (i.e., the number of attributes). The number of messages calculated here is an average number of messages that peers send to maintain their multiscale density within the whole simulation duration. We find that the number of messages needed is approximately doubled if the size of the network is doubled, while the dimensionality (the number of attributes) of the data tuple has little influence on the communication cost. As amortized, a lot of communication cost is saved since updates within the same scales share common update messages. Since each node maintains the multiscale densities and needs to communicate with other peers for any update, consequently more update messages are needed for larger sized networks.

Fig. 6. Effect of dynamic tuples.
The dimensionality of data has little influence, since the scale density computation in lower levels has taken place more on local peers. The number of scale levels computed on local peer is further determined by the size of the network. If peers in the network are sparser (i.e., more grids are maintained on one peer), then scales at lower levels will be more likely to be computed on local peers (i.e., less number of messages are needed).

Search precision is defined as the ratio of the top-k tuples over all fetched tuples in one personalized query evaluation. As we have known, the data space is divided into scales of different levels. All data tuples located in a scale are encoded with the same index key (or id). In our system, if a scale is overlapped with the search range (i.e., hyper-ellipsoid), all data tuples inside the scale are fetched. Hence, all the tuples in the scales overlapping with the border of the dedicated search range (see Fig. 5) will be fetched, but some of them are not to be ranked as the top-k (i.e., our algorithm contains false positive). These tuples will removed after the top-k ranking in terms of the user’s preference, and consequently resulting in the search precision evaluation. Fig. 6b shows the influence of dimensionality on search precision. As shown in Fig. 6b, the precision decreases as the dimensionality increases. It is because when the dimensionality increases, the number of partitioned intervals on each dimension is reduced. Therefore, more tuples will be located in one scale, resulting in the decrease of search precision.

Fig. 6c shows the influence of different data distributions on search precision, where the dimensionality is set to be 6 and the network size is 6000. In this experiment, normal data distribution is set with mean $\mu = 0$ and standard variation $\sigma = 1$. Data is distributed within the range $[-1000, 1000]$ for uniform distribution. The standard parameter is $\lambda = 0.001$ for exponential distribution, and $k = 1000$, $\alpha = 0.99$ for zipf distribution. As we can see, search precision on exponential and zipf distributions are lower than those on normal and uniform distributions. As for exponential and zipf distributions, data are concentrated within a small range, resulting in fetching a large number of data tuples in the frequently accessed scales, which degrades their search precision.

Fig. 6d shows the influence of coefficient of variations on search precision, where the network size is 6000 and the dimensionality is set to be 6. As shown in the figure, the search precision slightly decreases when the coefficient of variation of
personalization enlarges. This is because when the coefficient of variation becomes larger, the hyper-ellipsoid becomes flatter (i.e., more elliptical), leading to the fact that more scales are overlapped with the hyper-ellipsoid. Therefore, the precision decreases slowly.

6.1.2. Dynamic peers

In this set of experiments, peers may join, leave, or fail frequently, which lead to their managed data tuples to be inserted into or deleted from the network correspondingly and simultaneously. Networks with different preestablished maximum sizes are tested under the constraint that network sizes are never less than half of the maximum sizes within the whole simulation duration. We get similar results with the scenario of dynamic tuples, as shown in Fig. 7. The parameters of this set experiments are the same as those in scenarios of dynamic tuples.

Fig. 7a shows the communication cost for multiscale density construction. The communication cost is approximately doubled when the size of the network is doubled too. However, the communication cost is lower than that in the scenarios of dynamic tuples. Since many data tuples are updated simultaneously and may share a common update message (especially for those data tuples in the same peer), the communication cost is relatively smaller than that of dynamic tuples. Fig. 7b–d show similar results to those of scenarios of dynamic tuples. However, the search precision is comparatively lower because an update on a peer causes simultaneous updates on all its data tuples which may not be timely reflected in other peers.

6.2. A comparison study

As no general personalized query search method exists in P2P literature, we deploy the multi-step search [29] into the P2P environment as our baseline for comparison, where the network structure and the data index method are the same as those used in our system. For the multi-step search algorithm, it works in a filter-and-refine fashion. In the first stage, a quick $k$-nearest neighbor search on the index is performed to return the $k$ closest data tuples based on the coarser filter...
distance function. For those \( k \) tuples, their actual distances to the query are computed. In the second stage, a range query on the index is performed to find all the tuples whose filter distances are less than the \( k \)th largest actual distance, followed by the final actual distance computations. In our P2P network, it uses the \( k \)th largest actual distance as the initial search radius. The key difference between our method and the optimal multi-step method is that the range computation in our method is based on the filter distance function while the multi-step method is based on the pre-fetched tuples based on the filter distance function. Since the search range in the multi-step method is a hyper-sphere which contains the query hyper-ellipsoid, many redundant data tuples may be included and accessed, which costs much more.

Fig. 8 shows the comparison results with such a multi-step method on communication cost, search latency, search precision with various coefficients of variations and dimensionalities. In this experiment, by default, the number of peers is 10000, and the dimensionality is 6.

Fig. 8a shows the comparison on communication cost for personalized search on average, where the cost for multiscale density maintenance is not taken into account. Fig. 8b shows the comparison on search latency. As the multi-step method has no information on the data distribution in the whole data space and also uses a large initial search radius, it takes much longer than our method since more tuples are accessed. When the coefficient of variation of the personalized query is higher, the hyper-ellipsoid is flatter. Therefore, the scales overlapped with the hyper-ellipsoid are located on more peers, resulting in higher communication cost and higher search latency. However, the coefficient of variation affects the multi-step method much more significantly than ours. The comparisons on communication cost and latency are given in Fig. 8a and b. Clearly, our method outperforms the multi-step method greatly, especially when the coefficient of variation is large.

Fig. 8c shows the comparison on search precision as the coefficient of variation varies. As the estimated search radius in the multi-step method is based on pre-fetched data tuples, its estimation accuracy is not assured in the dynamic P2P environment. As a result, more undesired data tuples may be fetched for flatter hyper-ellipsoids in the multi-step method. Clearly, from Fig. 8c, our method outperforms the multi-step method significantly, and the improvement gaps get greater as the coefficient of variation goes up.

Fig. 8d shows the comparison on search precision as the dimensionality varies, where the coefficient of variation is set to be 2. As the dimensionality increases, the precision of both methods drop. This is reasonable since higher dimensionality leads to lower estimation accuracy. Nonetheless, our method still outperforms the multi-step method. Our method still achieves satisfactory precision (>80%) when the dimensionality reaches 6, while the multi-step method is only able to achieve about 50% precision.

In short, based on the above experiments, our method for personalized query evaluation in dynamic P2P networks is confirmed to be very effective and efficient.

7. Conclusion

In this paper, we present a novel idea for the evaluation of personalized query in P2P networks. Our method achieves high performance by transforming a personalized query into a range query based on accurate data density estimation by multiscale density estimation and matrix mapping by SVD. Detailed theoretical analysis, together with extensive experiments prove that our methods are highly efficient and effective for personalized query evaluation in dynamic P2P systems.

Acknowledgements

This work is partially supported by Shanghai Leading Academic Discipline Project No. B412, National Science Foundation of China under Grant Nos. 61003069 and 60925008, National 863 Program under Grant No. 2012AA011003.

References
