6-2019

Distributed similarity queries in metric spaces

Keyu YANG
Zhejiang University

Xin DING
Zhejiang University

Yuanliang ZHANG
Zhejiang University

Lu CHEN
Aalborg University

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

See next page for additional authors

Data Science and Engineering (2019) 4:93–108
https://doi.org/10.1007/s41019-019-0095-7

Distributed Similarity Queries in Metric Spaces

Keyu Yang[1] · Xin Ding[1] · Yuanliang Zhang[1] · Lu Chen[2] · Baihua Zheng[3] · Yunjun Gao[1]

Received: 3 June 2019 / Accepted: 12 June 2019 / Published online: 28 June 2019

Abstract
Similarity queries, including range queries and k nearest neighbor (kNN) queries, in metric spaces have applications in many areas such as multimedia retrieval, computational biology and location-based services. With the growing volumes of data, a distributed method is required. In this paper, we propose an Asynchronous Metric Distributed System (AMDS), to support efficient metric similarity queries in the distributed environment. AMDS uniformly partitions the data with the pivot-mapping technique to ensure the load balancing, and employs publish/subscribe communication model to asynchronous process large scale of queries. The employment of asynchronous processing model also improves robustness and efficiency of AMDS. In addition, we develop efficient similarity search algorithms using AMDS. Extensive experiments using real and synthetic data demonstrate the performance of metric similarity queries using AMDS. Moreover, the AMDS scales sublinearly with the growing data size.

Keywords Similarity query · Range query · kNN query · Metric space · Distributed processing · Algorithm

1 Introduction

Similarity queries in metric spaces find objects similar to a given query object under a certain criterion. This functionality has been widely used in real-life applications. This is because metric spaces can support various data types (e.g., images, words, DNA sequences) and flexible distance metrics (e.g., L_p-norm distance, edit distance). Here, we give two representative examples below.

Application 1 (Multimedia Retrieval) In an image retrieval system, the similarity between images can be measured using L_p-norm metric, earth mover's distance or other distance metrics between their corresponding feature vectors. Here, similarity queries in metric space can help users find similar figures for a given one.

Application 2 (Nature Language Processing) In the *WordNet*, a knowledge graph for better nature language understanding, the similarity between two words could be measured by the shortest path, maximum flow or other distance

Extended from their APWEB-WAIM 2018 papers

This journal paper is an extended version of the conference paper, titled "*Distributed k- Nearest Neighbour Queries in Metric Spaces*," which has been published in APWeb-WAIM 2018. Specifically, the paper extends the conference paper by including (i) one additional interesting query, i.e., metric range queries (Sect. 5.1); (ii) an enhanced experimental evaluation that incorporates the new class of queries and the flexibility of the system (Sect. 6); and (iii) more comprehensive coverage of related work (Sect. 2). Also, the introduction (Sect. 1) has been revised and additional theoretical analysis is included.

✉ Yunjun Gao
gaoyj@zju.edu.cn; raygaoyj@gmail.com

Keyu Yang
kyyang@zju.edu.cn

Xin Ding
dingxin@zju.edu.cn

Yuanliang Zhang
yuanlz@zju.edu.cn

Lu Chen
luchen@cs.aau.dk

Baihua Zheng
bhzheng@smu.edu.sg

[1] College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China

[2] Department of Computer Science, Aalborg University, Aalborg, Denmark

[3] School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902, Singapore

metrics. Here, similarity search in metric space can help users find the words that are closely related to a given one.

With the development of Internet, especially the widespread use of mobile devices, distributed data processing becomes a booming area in both the data management industry and academia [1–5]. Nowadays, the volume, richness and diversity of data challenges traditional metric similarity query processing in both space and time. This calls for a scalable metric similarity query method to provide efficient query service. Hence, in this paper, we investigate the distributed similarity queries in metric spaces.

Existing works on distributed processing in metric spaces [6–17] aim to accelerate similarity queries in parallelism and try to build a suitable network topology to manage the large amount of data. However, these existing solutions are not sufficient due to two reasons below. First, the ability to process a large quantity of similarity queries at the same time is in need nowadays. For example, an online image retrieval system (e.g., Google, Flickr) needs to provide the image search service to huge users at the same time. Second, the load balancing is also a basic need for distributed systems [18–20]. In particular, the load (e.g., the storage for data indexing and/or computational for query processing) across each node in the distributed system should be approximated the same. Motivated by these, we try to develop a distributed similarity query processing system in metric spaces that aims at efficient query processing in large scale and also taking the load balancing into consideration.

In order to design such a system, three challenges need to be addressed. The first challenge is *how to ensure the load balancing of the distributed system*? To ensure the load balancing, we uniformly divide the data into non-joint fragments using the pivot-mapping technique, and then distribute each fragment to a computational node of the system. The second one is *how to efficiently process metric similarity queries in large scale*? We utilize publish/subscribe communication mode to support synchronous process large scale of queries. Hence, massive queries can be executed with little time loss in message passing and the system can receive the quick response. The third challenge is *how to reduce the cost of a single similarity query*? We develop several pruning rules based on minimum bounding box (MBB) to avoid unnecessary calculations. In addition, the estimation-based kNN method is employed to further improve the efficiency of kNN queries. Based on these, we develop the Asynchronous Metric Distributed System (AMDS), to support efficient metric similarity queries in the distributed environment. To sum up, the key contributions of this paper are as follows:

- We present a pivot-mapping-based data partition method, which first uses a set of effective pivots to map the data from metric spaces to vector space, and then uniformly divides the mapped objects into disjoint fragments.
- We utilize the publish/subscribe communication model to asynchronously exchange messages without time wasting in network interactions, and thus to support large scale of similarity queries in metric spaces simultaneously.
- We propose a unified method to handle distributed similarity queries in metric spaces, where MBB is used to avoid redundant calculations.
- Extensive experiments using real and synthetic data evaluate the efficiency of AMDS and the performance of distributed similarity queries in metric spaces using AMDS.

The paper extends a preliminary study [21]. The extensions include support for the efficient processing of (i) one additional interesting query, i.e., metric range queries (Sect. 5.1); (ii) an enhanced experimental evaluation that incorporates the new class of queries and the flexibility of the system (Sect. 6); and (iii) more comprehensive coverage of related work (Sect. 2). Also, we have revised the introduction and have contained additional theoretical analysis.

The rest of this paper is organized as follows. Section 2 reviews related works. Section 3 introduces the definitions of metric similarity queries and the publish/subscribe communication mode. Section 4 elaborates the system architecture. Section 5 presents efficient metric similarity search algorithms. Experimental results and findings are reported in Sect. 6. Finally, Sect. 7 concludes the paper with some directions for future work.

2 Related Work

In this section, we review briefly related work on distributed metric similarity queries in Euclidean space and in metric spaces.

2.1 Euclidean Distributed Similarity Queries

Distributed similarity queries in Euclidean space have attracted a lot of attention since they are introduced. CAN [22] and Chord [16] build on top of DHT overlay network. LSH forest [23] uses locality-sensitive hash function to index data and perform (approximate) similarity queries on an overlay network. SWAM [24] consists of a family of distributed access methods for efficient similarity queries, which achieves the efficiency by bringing nodes with similar content together. DESENT [12, 25] is an unsupervised approach for decentralized and distributed generation of semantic overlay networks (SON). BATON [26] is a balanced tree structure on a peer-to-peer network and is capable of supporting both exact queries and range queries efficiently. VBI-Tree [27] is an abstract tree structure on top of an overlay network, which utilizes extensible centralized

Table 1 Frequently used notation

Notation	Description
O or P	A set of objects or pivots
o or p	An object or a pivot
$\phi(o)$	The vector for o mapped using P
MBB.lower(i)	Lower bound of MBB in dimension i
MBB.upper(i)	Upper bound of MBB in dimension i
MRQ(q, r)	A metric range query with query object q and query radius r
MkNNQ(q, k)	A metric kNN query with query object q and k
RR(q, r)	The query region of range query MRQ(q, r)
ND_k	The distance from the query object to its k-th nearest neighbor
lND_k	The estimation distance from the query object to its local k-th nearest neighbor
$\deg_{\mathrm{wp}}$	Degree of worker peers, the number of worker peers that master peer connects to
$\deg_{\mathrm{mp}}$	Degree of master peers, the number of master peers that root peer connects to
$\mathrm{num}_{\mathrm{mp}}$	Number of master peers
$\mathrm{num}_{\mathrm{wp}}$	Number of worker peers

mapping methods. Mercury [28] is proposed to support multiple attributes as well as explicit load balancing. P-Tree [10] is proposed to support range queries in addition to equality queries. NR-Tree [29] is a P2P adaption of R*-Tree [19], which supports range and kNN queries. In [30, 31], Skip Graphs [32] based system deals with load balancing for range queries. Range Guard Ring is used to optimize range queries by taking peer heterogeneity into consideration [33]. P-Ring [34], supporting equality and range queries, is fault-tolerant and skew data tolerant. FuzzyPeer [35] uses "frozen" technique to optimize query execution. A general and extensible framework in P2P network builds on the concept of Hierarchical Summary Structure [20]. Range query in tree-structured DHT is studied in [36]. More recently, VITAL [37] employs super-peer structure to exploit peer heterogeneity. However, all these above solutions focus on the vector space, which are unsuitable for metric distributed similarity queries. This is because, they utilize the geometric properties (e.g., locality-sensitive function [23], minimum bounding box [19]), to distribute the data on the underlying overlay network and to accelerate the query processing, which are unavailable in metric spaces.

2.2 Metric Distributed Similarity Queries

Existing methods for distributed similarity queries in metric spaces can be partitioned into two categories. The first category utilizes basic metric partitioning principles to distribute the data over the underlying network. GHT* and VPT* [7] use ball and generalized hyperplane partitioning principles, respectively. Besides GHT* and VPT*, efficient peer splits based on ball and generalized hyperplane partitioning techniques are also investigated in [11]. The second category utilizes the pivot-mapping technique to distribute the data. MCAN [38], relying on an underlying structured P2P network named CAN [22], maps metric data to vectors in an multi-dimensional space. M-Chord [39], relying on another underlying structured P2P network named Chord [16], uses iDistance [40] to map data into one-dimension values. M-Index [15] also generalizes iDistance technique to provide distributed metric data management. SIMPEER [41] works in autonomous manner, and uses the generated clusters by the iDistance method to further summarize peer data at supper peer level. An extension of SIMPEER is presented in [42], which focuses on recall-based range queries. In this paper, we adopt the pivot-mapping-based method due to two reasons below. First, pivot-mapping-based methods outperform metric partitioning based ones in terms of the number of distance computations [18, 43], one important criterion in metric spaces. Second, MCAN and M-Chord utilizing the pivot-mapping perform better than GHT* and VPT* using metric partitioning techniques [8, 9].

Apart from these, two general frameworks for metric distributed similarity search are proposed. One, called MESSIF, is an implementation framework with code reusing of GHT*, VPT*, MCAN, M-Chord, Chord and Skip Graphs [44]. Another framework utilizes a super-peer architecture and can support any underlying metric index in each peer, where super peers are responsible for query routing [17].

However, all these above methods are not enough due to two reasons below. First, they cannot support synchronous process large scale of metric similarity queries simultaneously, which is our focus. To address it, we utilize publish-subscribe communication model. Second, they do not take the load balancing into consideration, which is also important for distributed environment. To ensure the load balancing, we develop a pivot-mapping-based method to distribute data uniformly among the computational nodes.

3 Preliminaries

In this section, we review the metric space, similarity queries in metric spaces and publish-subscribe system. Table 1 summarizes the symbols frequently used throughout this paper.

3.1 Metric Similarity Queries

A metric space is denoted by a tuple (M, d), in which M is an object domain and d is a distance function to measure "similarity" between objects in M. In particular, the distance function d has four properties: (1) symmetry: $d(q, o) = d(o, q)$; (2) nonnegativity: $d(q, o) \geq 0$; (3) identity: $d(q, o) = 0$ iff $q = o$; and (4) triangle inequality: $d(q, o) \leq d(q, p) + d(p, o)$. Based on the properties of the metric space, we define two classes of similarity queries, i.e., range queries and k nearest neighbor (kNN) queries, in metric spaces, below.

Definition 1 *(Metric range query)* Given an object set O, a query object q, and a search radius r in M, a metric range query finds objects from O with their distances to q are bounded by r, i.e., $\text{MRQ}(q, r) = \{o | o \in O \wedge d(q, o) \leq r\}$.

Definition 2 *(Metric KNN query)* Given an object set O, a query object q, and an integer k in M, a metric kNN query finds k most similar objects from O for q, i.e., $MkNNQ(q, k) = \{R | R \subseteq O \wedge |R| = k \wedge \forall o_i \in R, \forall o_j \in O - R : d(q, o_j) \geq d(q, o_i)\}$

Consider an English word set O = {"defoliates," "defoliated," "defoliating," "defoliation," "citrate"}, where the edit distance is used to measure the similarity between two words. An example of range query is that finding the words from O with their (edit) distances to "defoliate" bounded by 1. The query result set MRQ("defoliate," 1) = {"defoliates," "defoliated"}; and an example of kNN query is that finding 2 most similar words for "defoliate," and the query result set $MkNNQ$("defoliate," 2) = {"defoliates," "defoliated"}. Note that, $MkNNQ(q, k)$ may be not unique, due to the distance tie. Nonetheless, the target of our proposed algorithms is to find one possible instance.

3.2 Publish/Subscribe

Publish-subscribe systems, also called as distributed event-bases systems [14], are systems where publishers publish structured events to an event service and subscribers express interest in particular events through subscriptions [45]. Here, the interest can be arbitrary patterns over the structured events. Publish-subscribe systems are used in a wide variety of application domains, particularly in those related to the large-scale dissemination of events, such as financial information systems, monitoring systems and so on.

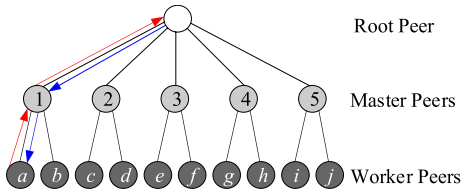


Fig. 1 Structure of AMDS

Publish-subscribe systems have two main characteristics: heterogeneity and asynchronicity. Heterogeneity means that components in a distributed system can work together as long as correct message is published and subscribed. Asynchronicity means publishers and subscribers are time-decoupled, message publishing and subscribing are performed independently. To sum up, the asynchronicity, the heterogeneity and the high degree of loose coupling suggest publish-subscribe systems perform well in dealing with large scale of messages simultaneously.

4 AMDS Architecture

In this section, we first give an overview of AMDS, and then present system organization and data deployment, respectively.

4.1 Overview

We develop AMDS, a three-layer tree structure on top of the overlay network, consisting three types of peers, root peer, master peers and worker peers, as depicted in Fig. 1. AMDS aims to answer a large-scale metric similarity queries in a distributed environment simultaneously. In the data deployment, AMDS divides the source data uniformly and distributes to worker peers, in order to achieve the load balancing. Here, any metric index can be used to index data in each worker peer. After the data deployment, AMDS can process two types of metric similarity queries, including metric range queries and metric kNN queries. During the query processing, metric similarity queries are published by worker peers in a bottom-up pattern, and subscribed by other worker peers in a top-down pattern.

4.2 System Organization

AMDS has three types of peers, root peers, master peers and worker peers, as depicted in Fig. 1. Peers are physical entities that have calculating and communication abilities. They are organized to index objects and accomplish

metric similarity queries. Peers can be partitioned into two classes in general: (i) worker peers, which directly index data objects and perform metric similarity queries, (ii) root peers and master peers, which manage children peers and distribute metric similarity queries over the system.

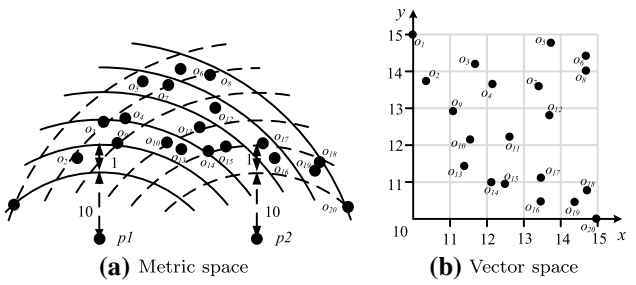
In AMDS, $\deg_w$ and $\deg_m$ are used to represent the number of worker peers (or master peers) that a master peer (or a root peer) connects to. The values of $\deg_w$ and $\deg_m$ depend on several factors including the network environment and the storage ability. Here, for simplify, one root peer is used and each master peer maintains equaled number of master peers in this paper. Hence, the value of $\deg_m$ equals to the number of master peers, and the value of $\deg_m \times \deg_w$ equals to the total number of worker peers. For example, as depicted in Fig. 1, $\deg_m = 5$ and $\deg_w = 2$. Usually the total number of worker peers is fixed. If $\deg_m$ becomes larger, more peer clusters will be managed by AMDS because all worker peers managed by a master peer form a peer cluster, and vice versa. Given a certain total number of peers, how the peers are organized into clusters will affect the overall performance of AMDS. In fact, the choice of cluster size is a trade-off, which a smaller clusters means less inner cluster communication cost and more inter cluster communication cost, and vice versa. Experiments are performed to verify the effect of cluster size setting.

In order to communicate between peers, we introduce the concept of missions. Missions are text messages exchanged among peers. Each metric similarity query can be packed into a mission. The mission issued by the worker peer is published to the corresponding master and then to the root peer. Every master peer (root peer) uses a mission list to maintain missions published by its children worker peers (master peers). Then, master peers (worker peers) subscribe the missions from the corresponding root peer (master peer).

In general, when a mission published, we do not know which worker peer will subscribe the mission. Hence, a mission usually contains (i) the mission ID, (ii) the mission type, (iii) the mission parameters, and (iv) the IP address of mission raiser, where the mission parameters include the query object q and query parameters R or k for metric range query or metric kNN query. For example, given a metric range query MRQ$(q, 5)$ issued by the worker peer whose IP address is 10.214.51.100, it can be packed into a mission as *"100, MRQ, q, 5, 10.214.51.100"*. Note that, the mission ID is a local identifier for a particular master peer (root peer), to avoid reprocessing missions.

4.3 Data Deployment

Assuming that worker peers have the same calculation ability and storage ability, we try to divide the source data equally among worker peers, to achieve the load balancing. After that, each worker peer can have its own objects



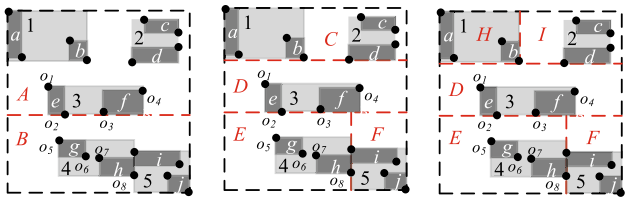
(a) Metric space **(b)** Vector space

Fig. 2 Pivot mapping

fragment. The framework of our data deployment is based on three phases, (i) the pivot-mapping of source data by root peers, (ii) the uniformly partitioning of data by master peers and (iii) the built of local index by worker peers.

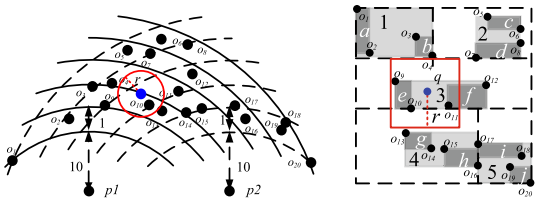
Pivot Mapping In the first stage, we map the objects in a metric space to data points in a vector space using well-chosen pivots. The vector space offers more freedom than the metric space when performing data partitioning and designing search approaches, since it is possible to utilize the geometric and coordinate information that is unavailable in the metric space. Given a pivot set $P = \{p_1, p_2, \ldots, p_n\}$, a general metric space (M, d) can be mapped to a vector space (R^n, L_∞). Specifically, an object o in a metric space is represented as a point $\phi(o) = \langle d(o, p_1), d(o, p_2), \ldots, d(o, p_n)\rangle$ in the vector space. For instance, consider the example in Fig. 2, where $O = \{o_i | 1 \le i \le 20\}$ and L_2-norm is used. If $P = \{p_1, p_2\}$, O can be mapped to a two-dimensional vector space, in which the x-axis represents $d(o_i, p_1)$ and the y-axis represents $d(o_i, p_2)$, $1 \le i \le 20$.

The selected pivots can drastically affect the performance of the search. It is shown that good pivots are always outliers, but outliers are not always good pivots [46]. Based on this observation, we develop our pivot selection algorithm. Our algorithm aims at finding out pivots far away from each other and from the rest of the objects in the database. In other words, the chosen pivots have to satisfy (i) they are outliers, and (ii) the distances between each other are as large as possible. Hence, our pivot selection algorithm is a two-stage based method. First, we use HF algorithm [13] to find a set of outliers, i.e., the pivot candidates. Then, the pivots are selected from the outlier set have the maximum sum of distances from each other. Here, the number of pivots to be selected is given and we fix the number of outliers at 40 (as in reference [37]), which is enough to find all outliers in our experiments. Note that, the HF algorithm does not need to take the whole object set as an input, and it works well using only a sampled object set. Moreover, theoretically, pivots do not need to be part of the object set. Consequently, objects can be inserted or deleted without changing the pivot set.



(a) First iteration **(b)** Second iteration **(c)** Third iteration

Fig. 3 Data partitioning



(a) Metric space **(b)** Vector space

Fig. 4 Metric range query

Data Partitioning Given a set of vectors after the pivot mapping, we first recursively partition the data objects into $\deg_m$ disjoined parts $P_i(1 \le i \le \deg_m)$ with equaled sizes, then further divide each P_i into $\deg_w$ disjoined equaled parts in a similar way. In particular, each time, we sort objects according to their values on a random selected dimension and then divided them into two non-intersected pieces in proportion $\lceil m/2 \rceil / \lfloor m/2 \rfloor$. Here, m denotes the number of master peers (or worker peers) to maintain these objects. Note that, the objects are partitioned into unequaled two parts when m is odd. For example, as depicted in Fig. 3, we sort objects $o(\in O)$ in the mapped vector space according to their values on dimension y. In the first partition iteration, the whole object set O can be partitioned into two parts A and B, where $|A| = 12, |B| = 8$, and $|A|/|B| = 3/2$ as $m = \deg_m = 5$. Here, $|A|$ and $|B|$ denote the number of objects in parts A and B, respectively.

First, the root peer publishes an initial data distributing mission. Then, master peers subscribe the data distribution mission. When recruiting mission is got, the *requester* will ask the mission *publisher* for data distribution. The *publisher* starts to distribute the objects by dividing them into two non-intersections and sending one intersection to the *requester*. New data distributing missions are published by master peers if the object fragments are still too large. This process will continue until all master peers get object fragments with suitable size according to their calculating and storage ability. For those master peers who have had finished the data distributing, they will continue to divide their own object fragments into equaled pieces and distribute to their children worker peers.

After the data partitioning, we build the local metric index on each worker peer. In this paper, we use M-Tree as default to index the objects distributed to the worker peer in the mapped vector space. Note that, any metric index is suitable to AMDS. In addition, a minimum bounding box (MBB) $M_i(= \{[a_i, b_i] | i \in [1, |P|]\})$ can be built for each worker peer. Specifically, a MBB M_i denotes the axis aligned *minimum bounding box* to contain all the mapped objects in the worker peer. Then, each master peer summarizes the MBB information from its children worker peers and formalizes a high-level MBB, to contain all the MBB of its children worker peers. Finally, the root peer summarizes all the MBB information from its children master peers.

Example 1 We give a running example of data distribution depicted in Fig. 3, where 20 objects are to be distributed in AMDS with 5 master peers and 10 worker peers. It is clear that, in this example, after the data partitioning, each worker peer will keep an index of 2 objects. First, the root peer publishes an initial data distribution mission. Master peer 1 gets the distribution mission, and then establishes a connection to root peer to get all the 20 objects. Since 20 is larger than 4 (= 20/5), the master peer 1 divides the 20 objects into two partitions A and B (as depicted in Fig. 3a), and then publishes a new data distribution mission. After that, master peer 4 gets the newly published distribution mission, establishes connection to master peer 1 to obtain 8 objects in area B. Since $|A|$ and $|B|$ are both larger than 4, master peers 1 and 4 continue to partition A and B, and two new distribution missions are published. This progress goes on until 5 master peers each obtain 4 objects. After the master peer obtains 4 objects, its children worker peers (e.g., a and b of m_1) are started and told connecting to master peer 1 to receive two objects from m_1 and builds local index on these two objects. Finally, MBB of each worker peer and master peer is built in bottom-up way.

5 Distributed Query Processing

In this section, we will introduce how to solve metric similarity queries in AMDS, range query and kNN query. First, the distribution of queries is introduced, and then the range query is introduced. Range query is the basic similarity query type processed by AMDS, and kNN query is introduced at last as an extended query based on range query.

5.1 Metric Range Query Processing

Metric Range queries can be regarded as a basic query type, in other words, other types of query (e.g., kNN queries) can be transformed into a range query. A metric range query retrieves the objects enclosed in the range region

that is an area centered at q with a radius r. Consider, for example, Fig. 4a, where a *circle* denotes a range region, and $\text{MRQ}(q, r) = \{o_2\}$. As discussed in Sect. 4, the objects are mapped into the vector space using a pivot set P to be indexed. Hence, the range region of $\text{MRQ}(q, r)$ can also be mapped into the vector space [10]. For instance, the red rectangle in Fig. 4b represents the *mapped range region* using $P = \{p_1, p_2\}$. To obtain $\text{MRQ}(q, r)$, we only need to verify the objects o whose $\phi(o)$ are contained in the mapped range region, as stated below.

Lemma 1 *Given a pivot set P, if an object o is enclosed in* $\text{MRQ}(q, r)$, *then* $\phi(o)$ *is certainly contained in the mapped range region* $\text{RR}(r)$, *where* $\text{RR}(r) = \{\langle s_1, s_2, \ldots, s_{|p|}\rangle \mid 1 \leq i \leq \mid P \mid \wedge s_i \geq 0 \wedge s_i \in [d(q, p_i) - r, d(q, p_i) + r]\}$.

Proof Assume, to the contrary, that there exists an object $o \in \text{MRQ}(q, r)$ but $\phi(o) \notin \text{RR}(r)$, i.e., $\exists p_i \in P$, $d(o, p_i) > d(q, p_i) + r$ or $d(o, p_i) < d(q, p_i) - r$. According to the triangle inequality, $d(q, o) \geq |d(q, p_i) - d(o, p_i)|$. If $d(o, p_i) > d(q, p_i) + r$ or $d(o, p_i) < d(q, p_i) - r$, then $d(q, o) \geq |d(o, p_i) - d(q, p_i)| - r$, which contradicts with our assumption. Consequently, the proof completes. □

According to Lemma 1, if the MBB of a worker peer wp_i or a master peer mp_i does not intersect with $\text{RR}(r)$, we can avoid performing $\text{MRQ}(q, r)$ on wp_i or mp_i. Considering the $\text{MRQ}(q, r)$ example depicted in Fig. 4, the master peer mp_4 does not need to perform $\text{MRQ}(q, r)$ as $M_4 \cap \text{RR}(r) = \emptyset$.

However, an important issue to be addressed is that how does the metric range query raiser know all the query results are returned. Multiple but not all the worker peers may have objects contained in the result of a metric range query, and it is impossible to known exact number in advance. In super-peer systems [41], all answers are collected by super peers and finally returned to query raiser by super peer. However, this pattern cannot work well in AMDS because super peers (e.g., the root peer) will be overloaded when tons of queries are needed to be processed. Hence, in AMDS, collecting answers is divided into two independent phases, counting the number of contributors and receiving answers from contributors. Here, a contributor denotes a worker peer whose MBB intersected with the mapped range region according to Lemma 1. The parent master peer of the worker peer that issues a metric range query is chosen to count the number of contributors. In particular, when a master peer *mp* receives a range query mission from its child worker peer wp, it counts the number of contributors, as *mp* maintains all the MBBs of its children worker peers. Then, the master peer publishes the mission to the root peer and receives the number of contributors returned by other master peers. When all the other master peers return the numbers, we sent the sum of these numbers to wp. Note that, the query raiser always knows the exact number of contributors of its raised query before it gets all returned answers, because in the distributing process, the counting step is performed by master peers that always earlier than local query processing step by worker peers.

Algorithm 1 MRQ_Publishing_WP Algorithm

Input: a metric range query $MRQ(q, r)$

```
1: if RR(r) ⊄ MBB then
2:     PublishMission(m) // compact MRQ(q, r) as a mission
3:     push m into the mission list L
4: else
5:     perform MRQ(q, r) on local worker peer
6: end if
```

We develop a MRQ_Publishing_WP Algorithm to publish a mission when a metric range query $\text{MRQ}(q, r)$ issued at worker peer wp_i, with the pseudocode depicted in Algorithm 1. The algorithm takes $\text{MRQ}(q, r)$ as an input. If MBB of wp_i does not contain $\text{RR}(r)$, which means other worker peers may have query result of $\text{MRQ}(q, r)$ due to Lemma 1, then the algorithm publishes a metric range query mission m to its parent master peer, and pushes m into the mission list L of wp_i (lines 1–3). Otherwise, the algorithm only needs to perform $\text{MRQ}(q, r)$ on wp_i.

Algorithm 2 Receiving_MP Algorithm

```
1: loop
2:     m = receiveMessage()
3:     if m is a metric range query mission then
4:         assignQuery(m, NULL) // push m into the mission list L
5:         if MBB ∩ m.RR(r) ≠ ∅ then
6:             m.answerCount = determineAnswers(m, childrenMBBList)
7:         end if
8:         if m.RR(r) ⊄ MBB then
9:             submitMission(m)
10:        end if
11:    else if m is an answer count message then
12:        mission = findMission(L, m.id)
13:        mission.answerCount += m.answers
14:        if all the master peers return the answer count message then
15:            send_message(mission.answerCount, mission.poster)
16:            mission.markAsFinished()
17:        end if
18:    end if
19: end loop
```

The master peer can receive messages from its children worker peers and other master peers. We develop Receiving_MP Algorithm to process such messages, with the pseudocode depicted in Algorithm 2. When the algorithm receives a message m, if m is a metric range query mission, according to Lemma 1, the algorithm calls assignQuery function to insert m into the mission list L and computes $m.answerCount$ (i.e., counts the number of contributors) if MBB of the maser peer intersects with the range region $\text{RR}(r)$ of the mission m, (line 3). Note that, NULL represents the destination of m is not known in advance. After that, if the range region $\text{RR}(r)$ is not contained in MBB, i.e., other worker peers not belonged to this master peer can also

contain the query result, then the algorithm calls publishToParent function to publish mission *m* to the root peer (lines 5–6). Otherwise, if *m* is an answer count message, then the algorithm calls findMission to find its corresponding mission, and adds *m.answers* to *mission.answerCount*. If all the master peers return the answer count messages, then it calls sentMessage to send *mission.answerCount* to *mission.poster* and mark mission as finished.

Algorithm 3 Subscribing_MP Algorithm

```
1: loop
2:     m = acquireMission()
3:     if MBB ∩ m.RR(r) ≠ ∅ then
4:         assignQuery(m, NULL)
5:     end if
6:     numOfAnswers = determineAnswers(m, childrenMBBList)
7:     sendMessage(numOfAnswer, m.submitter)
8: end loop
```

In addition, the master peer subscribes the root peer, and we develop a Subscribing_MP Algorithm, with its pseudocode depicted in Algorithm 3. When the algorithm receives a mission *m*, if *m.target* is itself or MBB intersects with the range region RR(*r*), then it calls assignQuery function to insert *m* into the corresponding mission list. After that, the algorithm counts the number of contributors (i.e., *numOfAnswer*) by calling determindeAnswers, and then sends *numOfAnswer* to *m.submitter*.

Algorithm 4 Receiving_WP Algorithm

```
1: loop
2:     m = receiveMessage()
3:     if m is an answer count messenger then
4:         mission = findMission(L, m.id)
5:         mission.numOfAnswer = m.numOfAnswer
6:     else if message is an result message then
7:         mission = findMission(L, m.id)
8:         R_s = R_s ∪ m.R_s
9:         if mission.numOfAnswer result messages received then
10:            mission.markAsFinished()
11:        end if
12:    end if
13: end loop
```

For any worker peer, it can receive messages from other worker peers. We develop a Receiving_WP Algorithm, with its pseudocode depicted in Algorithm 4. When the algorithm receives a message *m*, if *m* is an answer count message, it finds the corresponding mission *mission* in *L*, and sets the total number of contributors *mission.numOfAnswer*. If *m* is the result message, it finds the corresponding mission *mission* in *L*, and appends $m.R_s$ to the final result set R_s. If the number of the returned results reaches *mission.numOfAnswer*, it marks *mission* as finished.

Algorithm 5 Subscribing_WP Algorithm

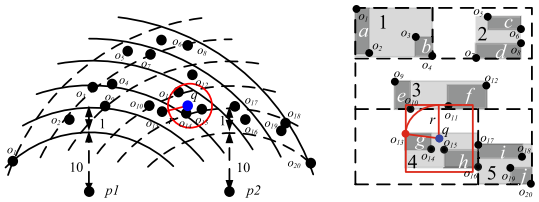
```
1: loop
2:     m = requireQuery()
3:     if m is a MRQ(q, r) mission then
4:         if MBB ∩ RR(r) ≠ ∅ then
5:             R_s = MBB(q, r) on local worker peer
6:             sendMessage(R_s, m.poster)
7:         end if
8:     end if
9: end loop
```

In addition, each work peer subscribes its parent master peer, and we develop a Subscribing_WP Algorithm, with its pseudocode depicted in Algorithm 5. When the algorithm receives a mission *m*, if *m* is a metric range query mission and MBB intersects with the range region RR(*r*), the algorithm performs local MRQ(q, r) on local worker peer and sends the result set R_s to *m.poster*.

Example 2 We illustrate distributed metric range query processing using the metric range query MRQ(q, r) depicted in Fig. 4. MRQ(q, r) is raised by worker peer wp_e, thus wp_e runs MRQ_Publishing_WP Algorithm. As RR(r) $\not\subset M_e$, wp_e publishes a mission *m* to its maser peer mp_3, and pushes *m* into the mission list *L* of wp_e. After that, Receiving_MP Algorithm running on master peer mp_3 receives the mission *m*. As *m* is a metric range query mission, and MBB M_3 intersects with RR(r), mp_3 pushes *m* into the mission List *L* of mp_3 and counts the number of contributors, i.e., $m.answerCount = 2$. In the sequel, as RR(r) $\not\subset M_e$, mp_3 publishes *m* onto the root peer. Master peers run Subscribing_MP Algorithm to subscribe the root peer. For example, mp_1 acquires the mission *m* from the root peer. As M_1 does not intersect RR(r), mp_1 returns the number of contributors (i.e., 0) to mp_3. At the same time, worker peers wp_e and wp_f run Subscribing_WP Algorithm to subscribe the master peer mp_3. Take wp_f as an example, it acquires the mission *m* from mp_3. Since MBB M_f intersects with RR(r), it performs the local MRQ(q, r) on wp_f and returns ∅ to wp_e. Finally, wpp_e runs Receiving_WP Algorithm to receive the result set sent by wp_f, and the mission terminates with the result set $\{o_2\}$ as all the worker peers have returned the result set.

5.2 Metric *k*NN Query Processing

A metric *k*NN query can be regarded as a metric range query with the search radius ND_k, where ND_k represents the distance from q to its k-th NN. However, ND_k is not known in advance, which makes metric *k*NN search is a little trickier than metric range retrieval. The metric *k*NN query can be solved using incremental metric range queries. More specifically, the search radius increases until k nearest neighbor



(a) Metric space **(b)** Vector space

Fig. 5 Metric kNN query

objects are retrieved [40, 47]. But in distributed network environment, this method is quite costly due to too many round-trips over the network. Hence, in this paper, we first estimate the value of ND_k and then perform a metric range query with at most two round-trips. However, it is difficult to obtain a good estimation of ND_k. An underestimation will cause extra round-trips, because the search radius needs to be enlarged to find kNN objects; an overestimation will cause extra filtering cost due to too much unnecessary verification of answers. Since one more round-trip will result in large communication cost, which is more costly, we need to obtain an overestimation and the estimation approaches the original value as much as possible. In order to obtain a good estimation of ND_k, we perform a local $MkNNQ(q,k)$ on the worker peer wp_i with minimum $MIND(MBB(\text{wp}_i), \phi(q))$ and use lND_k (i.e., the k-th nearest neighbor distance to q returned by local $MkNNQ(q,k)$) to estimate ND_k. This is because, the smaller $MIND(MBB(\text{wp}_i), \phi(q))$, the larger possibility that k NNs of q locates on wp_i.

Lemma 2 *Given a pivot set P, if an object o is enclosed in $MkNNQ(q,k)$, then $\phi(o)$ is certainly contained in the mapped range region* $\text{RR}(lND_k)$.

Proof Assume, to the contrary, that there exists an object $o \in MkNNQ(q,k)$ but $\phi(o) \notin \text{RR}(lND_k)$, i.e., $d(q,o) > lND_k$ due to the triangle inequality. However, according to the definition of lND_k, $lND_k \geq ND_k$, then $d(q,o) > ND_k$, which contradicts with our assumption. Consequently, the proof completes. □

Considering the example of Lemma 2 in Fig. 5, suppose that a local 2NN query is performed on worker peer wp_g and $lND_k = d(q, o_{13})$. According to Lemma 2, the result of 2NN query located in the mapped range region $\text{RR}(lND_k)$, as depicted in Fig. 5b.

Based on Lemma 2, the basic idea to perform metric kNN query processing in AMDS is depicted below. First, worker peer wp_i with the minimum $MIND(MBB(\text{wp}_i), \phi(q))$ is selected to perform a local kNN query and obtain lND_k, an estimation of ND_k. Then, $MkNNQ(q,k)$ is transformed into a metric range query $MRQ(q, lND_k)$, and published as a kNN mission $\{\}\}ID, MkNNQ, q, k, lND_k, poster'')$. Here, k is still needed because at most k objects will be sent back to the kNN query poster to reduce network volumes. Worker peers that receive such kNN missions will perform local range queries $MRQ(q, lND_k)$, and send at most k nearest objects of q to poster. When all contributors returned their results, the poster will find k NN objects in global as the final result.

Algorithm 6 kNN_Publishing_WP

Input: a metric kNN query $MkNNQ(q,k)$

```
1: if φ(q) ⊂ MBB then
2:     S_R = MkNNQ(q,k) and obtain lND_k
3:     submitMission(m) //m is a kNN query mission
4: else
5:     submitMission(m) //m is a local kNN query mission
6: end if
7: push m into the mission list
```

We develop a MkNN_Publishing_WP Algorithm to publish a mission when a metric kNN query $MkNNQ(q,r)$ issued at worker peer wp_i, with the pseudocode depicted in Algorithm 6. The algorithm takes $MkNNQ(q,r)$ as an input. If MBB of wp_i contains $\phi(q)$, then the algorithm performs a local kNN query on wp_i, obtains lND_k, and publishes a metric kNN query mission m to its parent master peer (lines 1–3). Otherwise, the algorithm submits a metric local kNN query mission m (lines 4–5). Finally, the algorithm pushes m into the mission list L of wp_i (lines 7).

Algorithm 7 kNN_Receiving_MP

```
1: loop
2:     m = receiveMessage()
3:     if m is a kNN request then
4:         lines 4-17 of Algorithm 2
5:     else if m is a local kNN request then
6:         if φ(q) ⊂ MBB then
7:             wp_i = findNearestChildren(m)
8:             assignQuery(m, wp_i)
9:         else
10:            submitQuery(m)
11:        end if
12:    end if
13: end loop
```

The master peer can receive messages from its children worker peers and other master peers. We develop Receiving_MP Algorithm to process such messages, with the pseudocode depicted in Algorithm 7. When the algorithm receives a message m, if m is a metric kNN query mission, the processing is similar as lines 4–17 of Algorithm 2. Otherwise, if m is a local kNN query mission and MBB contains $\phi(q)$, the algorithm calls findNearestChildren to find the worker peer wp_i with the minimum $MIND(MBB(\text{wp}_i), \phi(q))$ and push m into the mission list with m.target set to wp_i (lines 5–8). If

MBB does not contain $\phi(q)$, then the algorithm submits the mission m to the root peer.

Algorithm 8 kNN_Subscribing_MP (kSM)

```
1: loop
2:     m = acquireMission()
3:     if m is a local kNN mission then
4:         wp_i = findNearestChildren(m)
5:         assignQuery(m, wp_i)
6:     else if MBB ∩ RR(lND_k) ≠ ∅ then
7:         assignQuery(m, NULL)
8:         numOfAnswers = determineAnswers(m, childrenMBBList)
9:         sendAnswersTo(m.submitter, numOfAnswers)
10:    end if
11: end loop
```

In addition, the master peer subscribes the root peer, and we develop a kNN_Subscribing_MP Algorithm, with its pseudocode depicted in Algorithm 8. When the algorithm receives a mission m, if m is a local kNN mission, then it calls findNearestChildren to find the worker peer wp_i with the minimum $MIND(MBB(\text{wp}_i), \phi(q))$ and push m into the mission list with *m.target* set to wp_i (lines 2–5). Otherwise, if MBB intersects with RR(lND_k), then it pushes m into the mission list (lines 6–7). After that, the algorithm counts the number of contributors (i.e., *numOfAnswer*) by calling determindeAnswers, and then sends *numOfAnswer* to *m.submitter*.

Algorithm 9 kNN_Subscribing_WP

```
1: loop
2:     m = acquireMission()
3:     if m.target() is itself and m is a local kNN mission then
4:         S_R = MkNNQ(q, k) and obtain lND_k
5:         submitMission(mission, kNN)
6:     else if MBB ∩ RR(lND_k) ≠ ∅ then
7:         S_R = MkNNQ(q, k)
8:     end if
9:     send S_R to m.poster
10: end loop
```

For each worker peer, it can receive messages from other worker peers, and kNN_Receiving_WP Algorithm is the same as RQ_Receiving_WP Algorithm. In addition, each worker peer subscribes its parent master peer, and we develop a kNN_Subscribing_WP Algorithm (kSWA), with its pseudocode depicted in Algorithm 9. When kSMA receives a mission m, if *m.target* is itself and m is a local kNN mission, then it performs a local MkNN query to obtain lND_k, and submit a kNN mission (lines 3–5). Otherwise, if MBB intersects with RR(lND_k), then it performs a local MkNN (lines 6–7). After that, the algorithm sends the result set S_R to *m.poster* (line 9).

Example 3 An example of kNN query is shown in Fig. 5. Worker peer h raises a metric kNN query $MkNNQ(q, 2)$, but the query object q locates outside its MBB region, so h publishes a *hostedkNN* mission to master peer 4. Then, peer g is assigned to host this query by master peer 4 because q locates inside master peer 4 and g is nearest to q among children of master peer 4. Worker peer g performs an initial MkNN query and uses lND_2, the distance from q to its second nearest object o_5, marked as r in Fig. 5, to raise a kNN query mission like this, *"ID, MkNNQ, q, 2, r, h.address"*. It can be seen that the query region intersects with master peer 3 and 4, and worker peer e, f, h besides g. These peers perform local range query locally, and 5 objects are sent to h as the result, o_2, o_3, o_5, o_6 and o_7. After filtering step performed by query raiser h, the nearest two objects o_6 and o_7 are as the final answers.

5.3 Asynchronous Execution of Missions

In the network environment, the cost of network communication dominates in normal case. In order to achieve the query efficiency in the distributed environment, two rules should be considered: (i) the number of network communications should be decreased; and (ii) peers should not wait for communications with other peers. For the first philosophy, we develop Lemmas 1–2, to avoid unnecessary network communications for peers whose MBBs not intersected with the mapped range region. In addition, we develop a two-round MkNNQ algorithm to reduce the number of network communications. For the second philosophy, AMDS adopts the publish/subscribe model, which can support asynchronous execution of queries and thus can avoid waiting for communications with other peers. Based on these, AMDS can support efficient query processing in a large scale.

In AMDS, there are three types of characters during the query processing, i.e., the query raiser, the query broker and the query answerer. In particular, a query raiser is a peer which raises the query, a query answerer is a peer which performs the query and return the query answer to the query raiser, and a query broker is a peer that distribute the query to correct answerer. As discussed in Sects. 5.1 and 5.2, a query can be divided into four main phases, query raising, query distributing, query processing and result collecting. Each of these phases is processed by these characters independently, done by the query raiser, the query brokers, the query answerers and the query raiser, respectively. Obviously, four phases are loosely coupled, i.e., no strong relations between these phases exist, which is the premise of asynchronous execution.

Consider the example of asynchronous execution shown as Fig. 6, AMDS consists of two worker peers (i.e., x and y) and one master peer m. Each of the worker peers raises a query (i.e., q_1 and q_2) that both x and y related with the query. If two queries are processed in manner of asynchronous fashion, q_1 and q_2 are raised at the same time, and q_1 is distributed by m first, x will processes q_2 before y returns

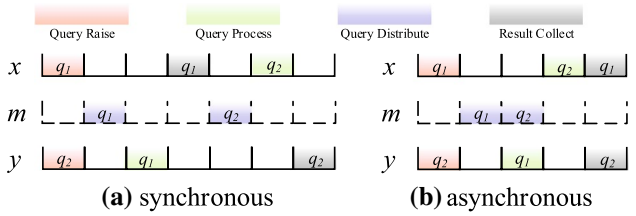


Fig. 6 Comparisons between execution modes

its answer to q_1. Although q_1 is finished earlier in Fig. 6a than that in Fig. 6b, it is obvious that asynchronous fashion is more efficient overall. Note that, the performance of synchronous fashion will get worse as the number of queries increases.

The asynchronous mode not only improves the query efficiency by reducing waiting time, but also brings enhancement in robustness. Consider, if a peer is down because of unpredictable error, and cannot provide results to all the queries it relates to. In synchronous pattern, the query processing is blocked by the first uncompleted query and the rest queries will remain unprocessed, but in asynchronous pattern, the query processing still keeps going, those queries affected by the down peer remain partial completed, but other queries will be processed normally. Hence, the influence of the down peers will be restricted only to their related queries, which enhances the robustness of AMDS.

6 Experimental Evaluation

In this section, we evaluate the effectiveness and efficiency of AMDS and metric similarity queries via extensive experiments, using both real and synthetic datasets. AMDS and corresponding metric similarity query algorithms are implemented in C++ with raw socket API. All the experiments are conducted on Intel E5 2620 processor and 64G RAM.

We employ two real datasets *Title*[1] and *CoPHIR*.[2] *Title* contains 800K PubMed paper titles, with strings whose length ranges from 8 to 666, resulting in an average length equaling to 71. Here, the similarity between two strings is measured using edit distance. *CoPHIR* consists of 1000K standard MPEG-7 image features extracted from Flickr, where the similarity between two features is measured as the L_2-norm. In addition, synthetic datasets *VECTOR* are generated with the cardinality range from 250K to 4M. Every dimension of *VECTOR* datasets is mapped to [0, 10,000]. Each *VECTOR* dataset has 10 clusters, and each cluster follows Gaussian distribution. Table 2 lists the statistics of the datasets used in our experiments.

Table 2 Statistics of three real-life datasets

Dataset	Cardinality	Dimension	Measurement
Title	800 K	36−666 (71 aver.)	Edit distance
CoPHIR	1000 K	40	L_2-norm
VECTOR	250 K−4M	8	L_∞-norm

Table 3 Parameter Settings

Parameters	Value
Cardinality	250K, 500K, 1M, 2M, 4M
The number of worker peers	1K, 2K, 4K, 8K, 16K
Query radius r(% of the maximum distance)	1%, 3%, 5%, 7%, 9%
k	1, 3, 9, 27, 81
The number of queries	4K, 12K, 20K, 28K, 36K

We investigate the performance of AMDS and metric similarity search algorithms under various parameters as summarized in Table 3. In each experiment, only one factor varies, whereas the others are fixed to their default values. For a fix number of worker peers, the number of master peers will affect the efficiency of AMDS and metric similarity queries. Hence, in the following experiments, the number of master peers is set as 32, 64 and 128 to evaluate the impact of the number of master peers, and 64 is set as default for synthetic datasets. The main performance metrics include (i) the CPU time and (ii) the network communication volume. Note that, in this paper, the number of pivots is set to 5.

6.1 Construction Cost

The first set of experiments verifies the AMDS construction cost, i.e., the cost of data deployment in AMDS. Here, the network communication volume is used as the performance metric. The result is demonstrated in Table 4. We collected the construction cost using both real and synthetic datasets. The number of worker peers is set to 4K as default, and three different sets of *number of master peers* are used. The first observation is that the data deployment in AMDS can achieve the efficiency in the network communication volume. This is because, as discussed in Sect. 4, the content of source dataset only copied twice in the data deployment process. It is first copied by root peer when passing data to master peers, and then copied by master peers when passing objects to worker peers. The second observation is that, the more master peers are, the higher construction cost is, and,

[1] http://www.ncbi.nlm.nib.gov/pubmed.

[2] http://cophir.isti.cnr.it/get.html.

Table 4 Construction cost of AMDS

Dataset	num_{mp}	Network communicate volume (KB)
Title	32	354,324
	64	397,376
	128	431,299
CoPHIR	32	1,804,711
	64	2,157,988
	128	2,240,842
VECTOR(250K)	64	200,954
VECTOR(500K)	64	398,878
VECTOR(1M)	64	792,620
VECTOR(2M)	64	1,587,004
VECTOR(4M)	64	3,174,662

the larger dataset is, the higher construction cost is. Because the network communication volume depends on the carnality of dataset and the topology of overlying network. In particular, with more master peer, more negotiations between master peers are needed, resulting in more network communication volume.

6.2 Evaluation of Metric Similarity Queries

The second set of experiments evaluates the performance of metric similarity queries using real and synthetic datasets. We study the influence of several parameters, including (i) the range radius r, (ii) the value k, (iii) the number of queries, (iv) the number of worker peers and (v) the cardinality of dataset.

Effect of r First, we investigate the performance of metric range queries using real datasets. The CPU time and the network communication volume of metric range queries are shown in Fig. 7 under various r values ranging from 1 to 9% of the maximum length. Note that, a default number of queries are set to 20K and each query object is generated randomly. Three lines presented in Fig. 7 represent 32, 64 and 128 master peers, respectively, as the number of worker peers is fixed at 4K. As observed, the query cost, including the CPU time and the network communication volume, increases with the growth of r. This is because, the search space grows as r increases, resulting in more related master peers and worker peers. In addition, AMDS using 32 master peers performs the best. Because the communication cost between master peers will reduce, the number of master peers decreases. However, the superiority of 32 master peers on *CoPHIR* is less distinct than that on *Title*. It is caused by the negative effects of reducing master peers. As the number of worker peers is fixed, the less master peers are, the more worker peers each master peer manage, resulting in more mission distribution cost among the cluster formed by a master peer. The dataset cardinality of *CoPHIR* is larger than *Title*, and the average length of each object is longer in *CoPHIR*. Hence, the negative effect counterbalanced the positive effect brought by reducing master peers, which makes the performance of 32 master peers not outstanding on *Title*.

Effect of k Second, we investigate the performance of metric kNN queries using real datasets. The CPU time and the network communication volume of metric range queries are shown in Fig. 8 under various k values ranging from 1 to 81. The first observation is that the query cost increases with the growth of k. This is because, the search space grows as k

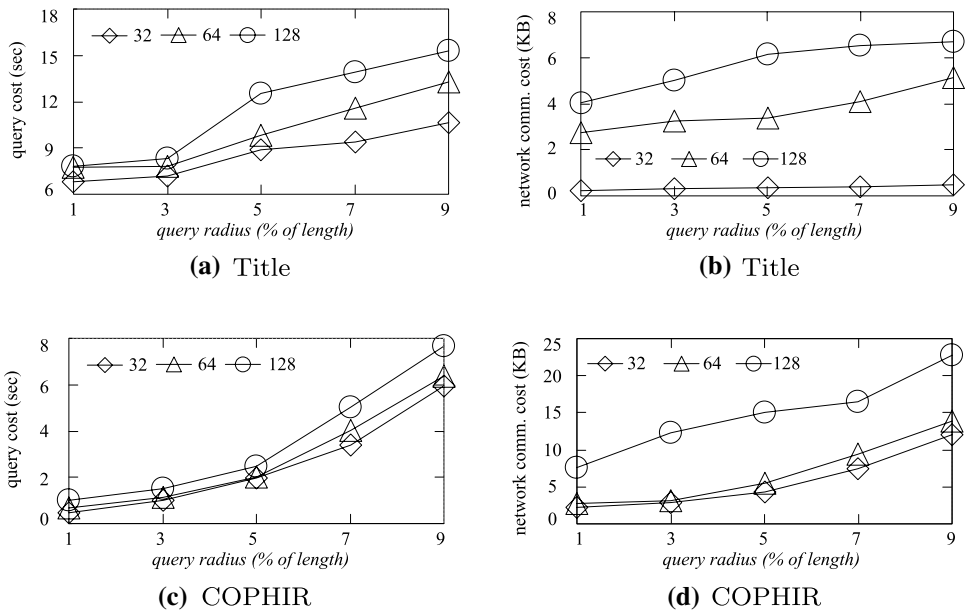


Fig. 7 Effect of query radius r

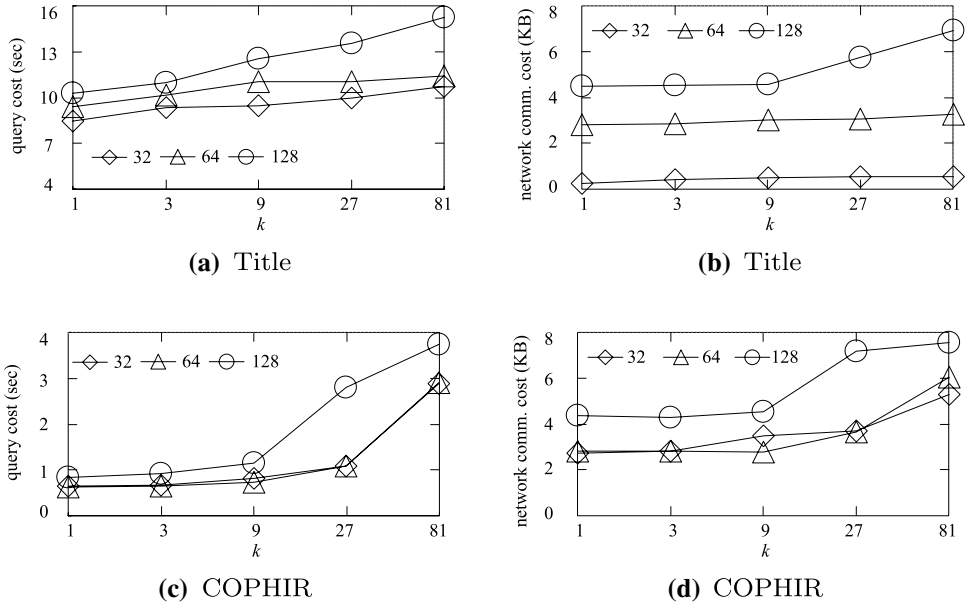


Fig. 8 Effect of k

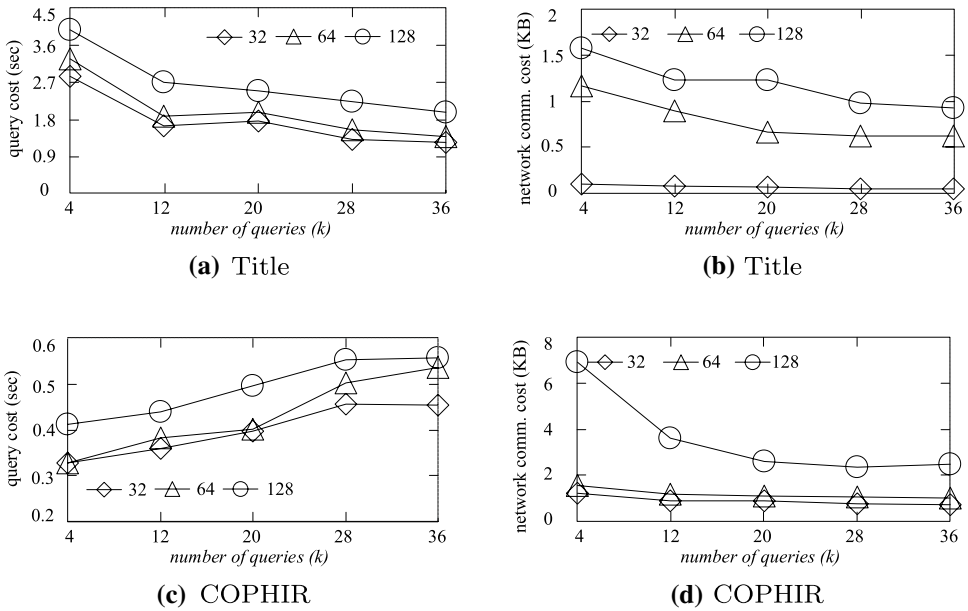


Fig. 9 Effect of number of queries

increases, resulting in more related master peers and worker peers. Note that, on *CoPHIR*, the CPU time and the network communication volume grow rapid when k exceeds 9 due to the distance distribution of the dataset.

Effect of Number of Queries Third, we explore the influence of number of queries on the efficiency of metric range queries using real datasets. Note that, we only use metric range queries to demonstrate the effect of number of queries due to space limitation and similar performance behavior on range queries. Figure 9 depicts the CPU time and the network communication volume of metric similarity queries. It is observed that in most cases the query cost decreases with the growth of the number of queries. This is because, although the total query cost increases, the average CPU time and network communication volume decrease due to the asynchronous executing mode of queries. However, on *CoPHIR*, the average CPU time ascends as the number of queries enlarges. The reason behind it is that, as discussed in Sect. 5.3, the average CPU time of asynchronous execution depends on the waiting time for communications with other peers. Compared with *Title*, *CoPHIR* needs less similarity query time on a single worker peer due to the simpler distance function L_2-norm used. But it needs more communication time because the average length of each object is longer than that of *Title*. Hence, *CoPHIR* has more waiting time for communication, resulting in a bad asynchronous execution

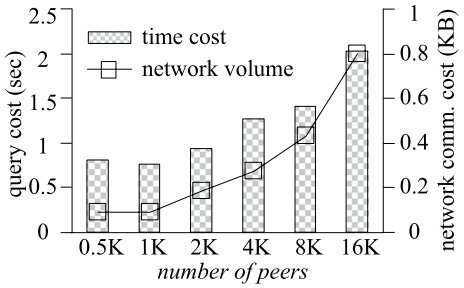


Fig. 10 Effect of num_{wp}

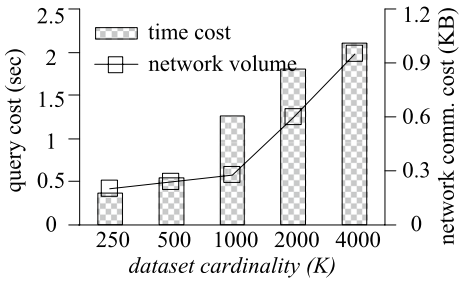


Fig. 11 Effect of cardinality

performance, and thus, an increment in the CPU time as the number of queries grows.

Effect of Number of Worker Peers Then, we evaluate the influence of number of worker peers. Figure 10 shows the results under various numbers of worker peers. The first observation is that the query cost first decreases from 0.5 to 1 K and then increases from 1K to 16K. This is because, with more worker peers, the objects managed by each worker peers get smaller, and thus, the metric similarity query cost on each worker peer decreases. However, at the same time, more time is consumed on the managing of a larger number of peers and communications between peers. In this case, 1K worker peers perform the best for *VECOTOR* on AMDS.

Effect of Cardinality After that, we study the impact of cardinality of using synthetic datasets, with the results depicted in Fig. 11. Here, we use 64 master peers as default. As expected, the query cost including CPU time and the network communication volume increases with the growth of cardinality.

Effect of Flexibility Finally, we verify that AMDS is flexible to support various overlying index structures. Here, three index structures are used, M-Tree, PM-Tree and linear scanning. Table 5 depicts the CPU time on real datasets, whereas the network communication volume is omitted, because the network cost will not change for different index structures. As expected, AMDS using PM-Tree performs the best, followed by AMDS using M-Tree and then linearly scanning, which keeps the consistency of the efficiency of the index structures. However, the index structures have little effect on the query efficiency in AMDS. The reason behind it is that the network communication cost is the dominate cost, as discussed in Sect. 5.3. In all the other experiments, we use M-Tree as the default index structure.

Table 5 Flexibility of AMDS

Dataset	Index structure	CPU cost (s)
Title	Linear scanning	2.62
	M-Tree	2.46
	PM-Tree	2.42
CoPHIR	Linear scanning	0.68
	M-Tree	0.50
	PM-Tree	0.49

7 Conclusions

In this paper, we present the Asynchronous Metric Distributed System (AMDS), which aims at dealing with a large scale of metric similarity queries. In the data deployment, AMDS uniformly partitions the data using the pivot-mapping technique for load balancing. Based on this, AMDS executes metric similarity queries in the form of missions and utilizes the publish/subscribe communication mode to support asynchronous processing and robustness. In addition, MBB technique is employed to further reduce the query cost, and each metric *k*NN query is solved using a local *k*NN query to estimate the *k*-th NN distance to avoid high network communication cost. Finally, extensive experiments on real and synthetic datasets show that the efficiency of AMDS and corresponding metric similarity search in both computational and communicational cost. In the future, we intend to use AMDS to support various metric queries, e.g., metric skyline queries.

Acknowledgements The preliminary version of this article has been published in *APWeb-WAIM* 2018 [21]. This work was supported in part by the National Key R&D Program of China Under Grant No. 2018YFB1004003, the 973 Program under Grant No. 2015CB352502, NSFC Grant No. 61522208 and NSFC-Zhejiang Joint Fund under Grant No. U1609217.

References

1. Vargas-Solar G, Zechinelli-Martini J, Espinosa-Oviedo J (2017) Big data management: what to keep from the past to face future challenges? Data Sci Eng 2(4):328–345
2. Chen B, Lv Z, Yu X, Liu Y (2017) Sliding window top-k monitoring over distributed data streams. Data Sci Eng 2(4):289–300
3. Zhao Z, Liu T, Li S, Li B, Du X (2017) Guiding the training of distributed text representation with supervised weighting scheme for sentiment analysis. Data Sci Eng 2(2):178–186
4. Onizuka M, Fujimori T, Shiokawa H (2017) Graph partitioning for distributed graph processing. Data Sci Eng 2(1):94–105
5. Zhang X, Chen L (2017) Distance-aware selective online query processing over large distributed graphs. Data Sci Eng 2(1):2–21
6. Batko M, Gennaro C, Zezula P (2005) A scalable nearest neighbor search in p2p systems. In: DBISP2P, pp 79–92
7. Batko M, Gennaro C, Zezula P (2005) Similarity grid for searching in metric spaces. In: Peer-To-peer, grid, and service-orientation in digital library architectures, pp 25–44
8. Batko M, Novak D, Falchi F, Zezula P (2008) Scalability comparison of peer-to-peer similarity search structures. Future Gener Comput Syst 24(8):834–848
9. Batko M, Novak D, Falchi F, Zezula P (2006) On scalability of the similarity search in the world of peers. In: International conference on scalable information systems, p 20
10. Crainiceanu A, Linga P, Gehrke J, Shanmugasundaram J (2004) P-tree: a p2p index for resource discovery applications. In: IEEE international conference on sensor networks, pp 390–391
11. Dohnal V, Sedmidubský J, Zezula P, Novák D (2008) Similarity searching: towards bulk-loading peer-to-peer networks. In: International workshop on similarity search and applications, pp 87–94
12. Doulkeridis C, Norvag K, Vazirgiannis M (2007) Desent: decentralized and distributed semantic overlay generation in p2p networks. IEEE J Sel Areas Commun 25(1):25–34
13. Filho RF, Traina AJ, Vieira MR, Faloutsos C (2007) The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. VLDB J 16(4):483–505
14. Mühl G, Fiege L, Pietzuch P (2006) Distributed event-based systems. Springer, Berlin
15. Novak D, Batko M, Zezula P (2012) Large-scale similarity data management with distributed metric index. Inf Process Manage 48(5):855–872
16. Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Comput Commun Rev 31(4):149–160
17. Vlachou A, Doulkeridis C, Kotidis Y (2012) Metric-based similarity search in unstructured peer-to-peer systems. Trans Large Scale Data Knowl Cent Syst V 5:28–48
18. Ares LG, Brisaboa NR, Esteller MF, Pedreira O, Places Ángeles S (2009) Optimal pivots to minimize the index size for metric access methods. In: International workshop on similarity search and applications, pp 74–80
19. Beckmann N, Kriegel HP, Schneider R, Seeger B (1990) The R⋆-tree: an efficient and robust access method for points and rectangles. In: SIGMOD, pp 322–331
20. Shen HT, Shu Y, Yu B (2004) Efficient semantic-based content search in p2p network. TKDE 16(7):813–826
21. Ding X, Zhang Y, Chen L, Gao Y, Zheng B (2018) Distributed *k*-nearest neighbor queries in metric spaces. In: APWeb-WAIM, pp 236–252
22. Ratnasamy S, Francis P, Handley M, Karp RM, Shenker S (2001) A scalable content-addressable network. In: SIGCOMM, pp 161–172
23. Bawa M, Condie T, Ganesan P (2005) Lsh forest: self-tuning indexes for similarity search. In: International conference on world wide web, pp 651–660
24. Banaei-Kashani F, Shahabi C (2004) Swam: A family of access methods for similarity-search in peer-to-peer data networks. In: ACM international conference on information and knowledge management, pp 304–313
25. Lua EK, Crowcroft J, Pias M, Sharma R, Lim S (2005) A survey and comparison of peer-to-peer overlay network schemes. IEEE Commun Surv Tutor 7(2):72–93
26. Jagadish HV, Ooi BC, Vu QH (2005) Baton: a balanced tree structure for peer-to-peer networks. In: VLDB, pp 661–672
27. Jagadish HV, Ooi BC, Vu QH, Zhang R (2006) Vbi-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes. In: ICDE, pp 34–34
28. Bharambe AR, Agrawal M, Seshan S (2004) Mercury: supporting scalable multi-attribute range queries. ACM SIGCOMM Comput Commun Rev 34(4):353–366
29. Liu B, Lee WC, Lee DL (2005) Supporting complex multi-dimensional queries in p2p systems. In: IEEE international conference on distributed computing systems, pp 155–164
30. Shu Y, Ooi BC, Tan KL, Zhou A (2005) Supporting multi-dimensional range queries in peer-to-peer systems. IEEE P2P:173–180
31. Ganesan P, Bawa M, Garcia-Molina H (2004) Online balancing of range-partitioned data with applications to peer-to-peer systems. In: VLDB, pp 444–455
32. Aspnes J, Shah G (2003) Skip graphs. ACM Trans Algorithm 3(4):37
33. Ntarmos N, Pitoura T, Triantafillou P (2007) Range query optimization leveraging peer heterogeneity in DHT data networks. In: Databases, information systems, and peer-to-peer computing, pp 111–122
34. Crainiceanu A, Linga P, Machanavajjhala A, Gehrke J, Shanmugasundaram J (2007) P-ring: an efficient and robust p2p range index structure. In: SIGMOD, pp 223–234
35. Kalnis P, Ng WS, Ooi BC, Tan K-L (2006) Answering similarity queries in peer-to-peer networks. Inf Syst 31(1):57–72
36. Datta A, Hauswirth M, John R, Schmidt R, Aberer K (2005) Range queries in trie-structured overlays. In: IEEE international conference on peer-to-peer computing, pp 57–66
37. Ghanem SM, Ismail MA, Omar SG (2015) Vital: structured and clustered super-peer network for similarity search. Peer-to-Peer Netw Appl 8(6):965–991
38. Falchi F, Gennaro C, Zezula P (2007) A content–addressable network for similarity search in metric spaces. In: DBISP2P, pp 98–110
39. Novak D, Zezula P (2006) M-chord: a scalable distributed similarity search structure. In: International conference on scalable information systems, p 19
40. Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) idistance: an adaptive b+-tree based indexing method for nearest neighbor search. ACM Trans Database Syst 30(2):364–397
41. Doulkeridis C, Vlachou A, Kotidis Y, Vazirgiannis M (2007) Peer-to-peer similarity search in metric spaces. In: VLDB, pp 986–997
42. Doulkeridis C, Vlachou A, Kotidis Y, Vazirgiannis M (2009) Efficient range query processing in metric spaces over highly distributed data. Distrib Parallel Databases 26(2–3):155–180
43. Chávez E, Navarro G, Baeza-Yates R (2001) Searching in metric spaces. ACM Comput Surv 33(3):273–321
44. Batko M, Novak D, Zezula P (2007) Messif: metric similarity search implementation framework. In: Digital libraries: research and development, pp 1–10
45. Coulouris GF, Dollimore J, Kindberg T (2005) Distributed systems: concepts and design. Pearson Education, London

46. Bustos B, Navarro G, Chávez E (2003) Pivot selection techniques for proximity searching in metric spaces. Pattern Recogn Lett 24(14):2357–2366

47. Yu C, Ooi BC, Tan KL, Jagadish HV (2001) Indexing the distance: an efficient method to knn processing. In: VLDB, pp 421–430