Distributed k-Nearest Neighbor Queries in Metric Spaces

Xin Ding[1], Yuanliang Zhang[1], Lu Chen[2], Yunjun Gao[1](✉), and Baihua Zheng[3]

[1] College of Computer Science, Zhejiang University, Hangzhou, China
{dingxin, yuanlz, gaoyj}@zju.edu.cn

[2] Department of Computer Science, Aalborg University, Aalborg, Denmark
luchen@cs.aau.dk

[3] School of Information Systems, Singapore Management University, Singapore, Singapore
bhzheng@smu.edu.sg

Abstract. Metric k nearest neighbor (MkNN) queries 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), which uniformly partitions the data with the pivot-mapping technique to ensure the load balancing, and employs publish/subscribe communication model to asynchronously process large scale of queries. The employment of asynchronous processing model also improves robustness and efficiency of AMDS. In addition, we develop an efficient estimation based MkNN method using AMDS to improve the query efficiency. Extensive experiments using real and synthetic data demonstrate the performance of MkNN using AMDS. Moreover, the AMDS scales sub-linearly with the growing data size.

Keywords: Metric space · k nearest neighbor query · Publish/subscribe Query processing · Algorithm

1 Introduction

Metric k nearest neighbor (MkNN) queries find k objects most similar to a given query object under a certain criterion. 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), this functionality has been widely used in real life applications. 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, MkNN queries in metric space can help users to locate figures that are similar as 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 metrics. Here, M*k*NN queries can help users to find the words that are closely related to a given one.

With the development of Internet, especially the widespread use of mobile devices, the volume, richness and diversity of data challenge the traditional M*k*NN query processing in both space and time. This calls for a scalable M*k*NN method to provide efficient query service. Hence, in this paper, we investigate the distributed M*k*NN queries.

Existing works on distributed processing in metric spaces [1–10] aim to accelerate M*k*NN queries in parallelism and try to build a suitable network topology to manage the large amount of data. However, the existing solutions are not sufficient because of following two main reasons. First, the ability to process a large quantity of M*k*NN queries simultaneously is in need nowadays. Second, the load balancing is also a basic need for distributed systems [11–13]. Motivated by these, we try to develop a distributed M*k*NN query processing system that takes the load balancing into consideration and aims at efficient query processing in large scale.

In order to design such a system, three challenges need to be addressed. The first challenge is how to ensure the load balancing of a distributed system? To ensure the load balancing, we uniformly divide the data into disjoint fragments using the pivot mapping technique, and then distribute each fragment to a computational node. The second one is how to efficiently process queries in large scale? To support synchronous process of large scale of queries, we utilize publish/subscribe communication model, and thus, massive queries can be executed with negligible time loss in message passing. The third challenge is how to reduce the cost of a single similarity query? We develop several pruning rules with the minimum bounding box (MBB) to save unnecessary verifications. In addition, an estimation based M*k*NN method is employed to further improve the query efficiency. Based on these, we develop the Asynchronous Metric Distributed System (AMDS) to support efficient M*k*NN queries in the distributed environment. To sum up, the key contributions in 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 a metric space to a vector space, and then uniformly divides the mapped objects into disjoint fragments.
- We utilize the publish/subscribe communication model to asynchronously exchange messages that saves time in network communication, and thus to support large scale of M*k*NN query processing simultaneously.
- We propose an estimation-based method to handle M*k*NN queries, where pruning rules with MBB are used to avoid redundant verifications.
- Extensive experiments using real and synthetic data evaluate the efficiency of AMDS and the performance of distributed M*k*NN queries using AMDS.

The rest of this paper is organized as follows. Section 3 reviews related works. Section 3 introduces the definitions of M*k*NN queries and the publish/subscribe communication model. Section 4 elaborates the system architecture. Section 5 presents an efficient algorithm for M*k*NN searches. Experimental results and findings are reported in Sect. 6. Finally, Sect. 7 concludes the paper with some directions for future work.

2 Related Work

We review briefly related work on distributed *k*NN queries in Euclidean and metric spaces.

2.1 Distributed Euclidean *k*NN Queries

Distributed *k*NN queries in Euclidean space have attracted a lot of attention since they are introduced. CAN [14] and Chord [9] build on top of DHT overlay network. LSH forest [15] uses a set of locality-sensitive hash functions to index data and perform (approximate) *k*NN queries on an overlay network. SWAM [16] consists of a family of distributed access methods for efficient *k*NN queries, which achieves the efficiency by bringing nodes with similar contents together. DESENT [17] is an unsupervised approach for decentralized and distributed generation of semantic overlay networks. VBI-Tree [18] is an abstract tree structure on top of an overlay network, which utilizes extensible centralized mapping methods. Mercury [19] is proposed to support multiple attributes as well as explicit load balancing. NR-Tree [20] is a P2P adaption of R*-Tree [12] to support *k*NN queries. FuzzyPeer [21] uses “frozen” technique to optimize query execution. A general and extensible framework in P2P network builds on the concept of hierarchical summary structure [12]. More recently, VITAL [22] employs a super-peer structure to exploit peer heterogeneity. However, all these above solutions focus on the vector space and they utilize the geometric properties (e.g., locality sensitive function [15], minimum bounding box [12]) that are unavailable in metric spaces, to distribute the data on the underlying overlay network and to accelerate the query processing. Hence, they are unsuitable for distributed M*k*NN queries.

2.2 Distributed M*k*NN Queries

Existing methods for distributed M*k*NN queries can be clustered into two categories. The first category utilizes basic metric partitioning principles to distribute the data over the underlying network. GHT* and VPT* [2] 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 [5]. The second category utilizes the pivot mapping technique to distribute the data. MCAN [23], relying on an underlying structured P2P network named CAN [14], maps data to vectors in a multi-dimensional space. M-Chord [24], relying on another underlying structured P2P network named Chord [9], uses iDistance [25] to map data into one-dimension values. M-Index [8] also generalizes iDistance technique to provide distributed metric data management. SIMPEER [6] works in autonomous manner, and uses the generated clusters obtained by the iDistance method to further summarize peer data at the supper peer level. In this paper, we adopt the pivot-mapping based method. This is because pivot-mapping based methods outperform metric partitioning based ones in terms of the number of distance computations [1, 26], one important criterion in metric spaces. As an example, MCAN and M-Chord utilizing the pivot mapping perform better than GHT* and VPT* using metric partitioning techniques [3, 4].

Apart from these, two general frameworks for distributed M*k*NN search are proposed. One, called MESSIF, is an implementation framework with code reusing of GHT*, VPT*, MCAN, M-Chord, Chord and Skip-Graphs [27]. The other utilizes a super-peer architecture, where super-peers are responsible for query routing [10].

However, all these above methods are not sufficient due to two reasons below. First, they cannot support synchronous processing of large scale of M*k*NN queries simultaneously, which is our main objective. To address it, we develop methods based on 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 partition method to distribute the data uniformly among the computational nodes.

3 Preliminaries

In this section, we review the M*k*NN queries and publish-subscribe system. Table 1 summarizes the symbols frequently used throughout this paper.

Table 1. Symbols and description

Notation	Description
O or P	A set of objects or pivots
o or p	An object or a pivot
$\phi(o)$	A vector for object o after pivot-mapping
wp_i or mp_i	A worker peer or a master peer
MBB(wp_i) or MBB(mp_i)	The minimum bounding box for wp_i or mp_i
m	A mission used for communication among peers
M*k*NN(q, k)	A metric k nearest neighbor query w.r.t. q and k
$d()$	The distance function in a metric space
$RR(q, r)$	A metric range region centered at q with radius r
$q.d_k$	The distance between q to its k-th nearest neighbor
$q.d_k^i$	An estimation of $q.d_k$
deg_w	The number of worker peers that a master peer connects to
deg_m	The number of master peers that a root peer connects to
num_{wp}	The total number of worker peers

3.1 M*k*NN 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) *non-negativity*: $d(q, o) \geq 0$; (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 these properties, we define M*k*NN queries.

Definition 1 (MkNN Query). *Given an object set O, a query object q, and an integer k in M, a MkNN query finds k most similar objects from O for q, i.e.,* $MkNN(q, k) = \{R \mid R \subseteq O \wedge |R| = k \wedge \forall o_i \in R, \forall o_j \in O - R: d(q, o_j) \geq d(q, o_i)\}$.

An M*k*NN query can be regarded as a metric range (MR) query if the *k*-th nearest neighbor distance if known in advance (i.e., the search radius), as defined below.

Definition 2 (MR Query). *Given an object set O, a query object q, and a search radius r in M, a metric range (MR) query finds objects from O with their distances to q are bounded by r, i.e.,* $MR(q, r) = \{o \mid o \in O \wedge d(q, o) \leq r\}$.

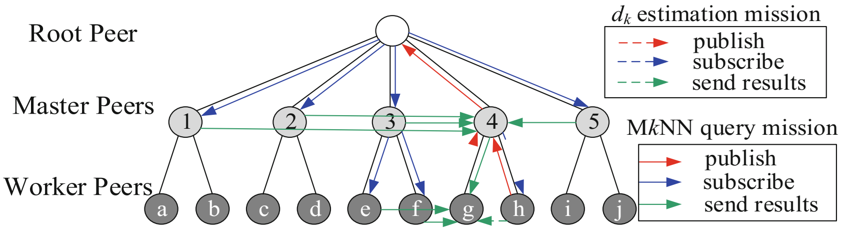


Fig. 1. AMDS structure and communications for M*k*NN processing

3.2 Publish/Subscribe

Publish/subscribe system, also termed as distributed event-bases system [28], is a system where publishers publish structured events to an event service and subscribers express interest in particular events through subscriptions [29]. 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 cooperative working systems where a number of participants need to be informed of events of shared interest. Hence, in this paper, we adopt the publish/subscribe communication model to support large scales of M*k*NN query processing simultaneously, which is required in real life applications.

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 that publishers and subscribers are time-decoupled, and message publishing and subscribing are performed independently. Hence, the asynchronicity, the heterogeneity, and the high degree of loose coupling suggest that publish/subscribe systems perform well in dealing with large scale of messages.

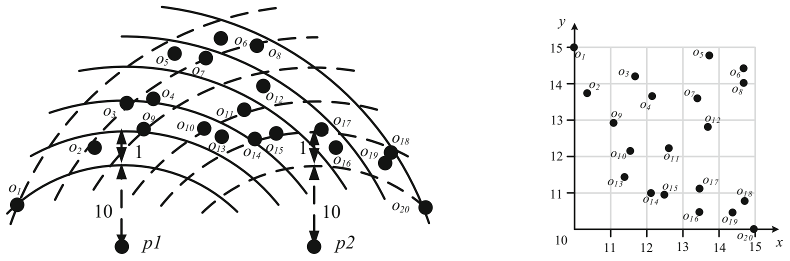
4 AMDS Architecture

In this section, we present system organization and data deployment of AMDS system.

4.1 System Organization

AMDS aims to answer a large scale of M*k*NN queries in a distributed environment simultaneously. In the following, we first introduce the system structure, and then describe the communications in the system.

System Structure. AMDS is a three-layer tree structure on top of the overlay network, consisting of three types of peers, termed as *root peers*, *master peers* and *worker peers*, as depicted in Fig. 1. Peers are physical entities with calculation and communication abilities. They are organized to index objects and to accomplish M*k*NN queries. Worker peers (e.g., wp_a, wp_b) directly index data objects and perform metric similarity queries locally; while root peers and master peers (e.g., mp_1, mp_2) manage children peers and distribute MkNN queries over the system.



(a) Metric space (b) Vector space

Fig. 2. Pivot mapping

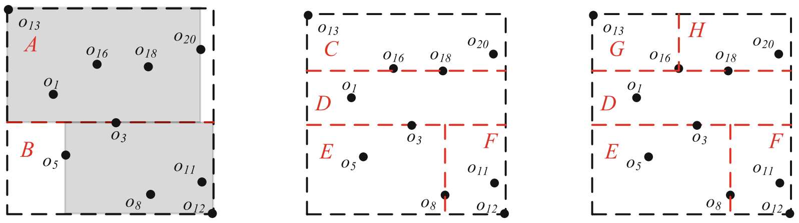
In AMDS, deg_w/deg_m represent the number of worker peers/master peers that a master peer/a root peer connects to, respectively. The values of deg_w and deg_m depend on several factors, including the network environment and the storage ability. For simplify, in this paper, we assume there is one root peer, and each master peer maintains an equal number of worker peers. 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, e.g., $deg_m = 5$ and $deg_w = 2$ for the example system depicted in Fig. 1. For clarity, we name a master peer with all the children worker peers as a *peer cluster*.

System Communication. To support communications between peers, we introduce the concept of *missions*. Missions are text messages exchanged among peers for communications. Data deployment, object updating operations, or MkNN queries can be packed into missions. The missions are published by worker peers in a bottom-up pattern, and subscribed by other worker peers in a top-down pattern, as illustrated in Fig. 1. More specifically, a worker peer, the owner of a mission, can publish a mission to its parent master peer and then to the root peer. Then, master peers can subscribe to the missions from the root peer and worker peers can subscribe to the missions from their master peers. Every master peer (or root peer) maintains a mission list to keep track of all the missions published by its children worker peers (or master peers).

4.2 Data Deployment

To achieve the load balancing, we divide the source data equally among worker peers, assuming that worker peers share the same calculation ability and storage capacity. Our framework of data deployment contains three phases, (i) the pivot-mapping of source data performed by root peers, (ii) the partitioning of mapped data performed by master peers, and (iii) the local index building performed 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 $O = \{o_i \mid 1 \leq i$ $20\}$ 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 $d(o_i, p_2)$ $(1 \leq i \leq 20)$. In particular, object o_1 is mapped to point $\langle 10, 15\rangle$.



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

Fig. 3. Sample-based data partitioning

The quality of selected pivots has a marked impact on the search performance. It is shown that good pivots are far away from each other and from the rest of the objects in the database [30]. Based on this observation, we select pivots in a way such that (i) they are outliers, and (ii) the distances between each other are as large as possible. Moreover, theoretically, pivots do not need to be part of the object set. Consequently, the quality of pivots is highly related with the data distribution and we have the flexibility to insert/delete objects without changing the pivot set.

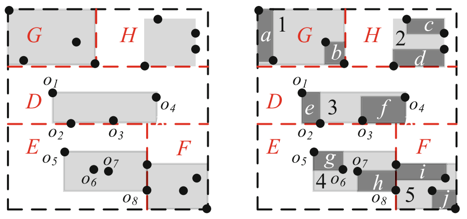
Data Partitioning. The root peer first samples the whole dataset and then maps the sampled data objects into a set of vectors using selected pivots as discussed above. After that, the root peer partitions the data objects into deg_m disjoint parts P_i $(1 \leq i$ $deg_m)$ of equivalent size, with $BB(P_i)$ representing the bounding box corresponding to each part P_i. Here, $BB(P_i)$ is an axis aligned bounding box and it contains the part P_i such that $\forall 1 \leq i < j \leq deg_m$, $BB(P_i) \cap BB(P_j) = \emptyset$ and $\cup_{1 \leq i \leq degm} BB(P_i)$ covers the entire space. Then, each $BB(P_i)$ ($1 \leq i \leq deg_m$) is assigned to the corresponding master peer mp_i.

We give an example of sample-based data partition in Fig. 3. As depicted in Fig. 3 (a), we sort sampled objects o ($\in O$) in the mapped vector space according to their values on dimension y. In the first iteration, based on $deg_m = 5$ and $\lceil deg_m/2 \rceil / \lfloor deg_m/2 \rfloor = 3/2$, we partition the whole sampled dataset into two parts $A = \{o_1, o_3, o_{13}, o_{16}, o_{18}, o_{20}\}$ and $B = \{o_5, o_8, o_{11}, o_{12}\}$. The partition continues until five equal parts are obtained, i.e., $D = \{o_1, o_3\}$, $E = \{o_5, o_8\}$, $F = \{o_{11}, o_{12}\}$, $G = \{o_{13}, o_{16}\}$ and $H = \{o_{18}, o_{20}\}$, with corresponding bounding boxes (i.e., the dotted rectangle) depicted in Fig. 3(c). In the sequel, each data object o is associated to the corresponding master peer m_i with $\phi(o) \in BB(P_i)$. In addition, the minimum bounding box (MBB), i.e., the light gray rectangles depicted in Fig. 4(a), is built for each master peer m_i accordingly. Specifically, a MBB(mp_i) denotes the axis aligned *minimum bounding box* to contain all the mapped objects in mp_i. After that, each master peer further divides each P_i into deg_w disjoined equaled parts in a similar way and MBBs are also built for all the worker peers. The dark gray rectangles, depicted in Fig. 4(b), represent the MBBs for worker peers.

Local Index Construction. Finally, each worker peer builds a local metric index for all its objects. Here, we use M-tree to index the objects distributed to each worker peer in the mapped vector space.

Algorithm 1 *k*NN_*WP*

```
Input: $MkNN(q, k)$ issued at $wp_i$
1: if $\phi(q) \subset MBB(wp_i)$ then
2:     $R$ := findkNN($wp_i$, $q$, $k$)
3:     $q.d_k^i := d(q, R[k])$
4:     $R$ := MR($q$, $q.d_k^i$, $wp_i$)
5: else
6:     $m$ := newMission($ID$, $DEst$, $q$, $IP_i$)
7:     sendMessage($wp_i$, $m$, IP($wp_i.parent$))
8:     $msg$ := receiveMessage()
9:     while ($msg.ID \neq m.ID$ and
               $msg.type \neq$ DEst)
10:        $msg$ := receiveMessage()
11:    $R$ := MR($q$, $msg.content$, $wp_i$)
12: return findkNN($R$, $q$, $k$)
```



(a) MBBs in master (b) MBBs in worker

Fig. 4. MBBs after data partitioning

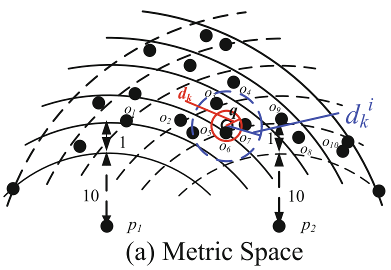
5 Distributed Query Processing

In this section, we present how to support M*k*NN queries in AMDS. We first introduce the algorithms to support metric range query and metric *k*NN query, and then present the asynchronous execution of missions.

5.1 M*k*NN Query Processing

Two solutions exist to answer M*k*NN query. One possible solution is incrementally increasing the search radius until *k* nearest neighbor objects are retrieved [25, 31]. However, in a distributed environment, this method incurs very expensive communication cost due to too many message exchanges over the network. Alternatively, AMDS adopts a different approach. It performs a metric range query based on an estimated search radius with at most two round-trips message exchanges.

A metric range query retrieves the objects enclosed in the range region that is an area centered at *q* with a radius *r*. The range region of *MR*(*q*, *r*) can also be mapped into the vector space [32]. Consider, for example, Fig. 5(a), where a *blue dotted circle* denotes a range region, and the *blue rectangle* in Fig. 5(b) represents the *mapped range region* using $P = \{p_1, p_2\}$. To obtain *MR*(*q*, *r*), we only need to verify the objects *o* whose $\phi(o)$ are contained in the mapped range region, as stated below.



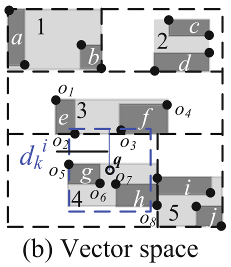


Fig. 5. M*k*NN query (Color figure online)

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

Proof. Assume, to the contrary, that there exists an object $o \in MR(q, r)$ but $\phi(o) \notin RR(r)$, 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, $d(q, o) \ge |d(q, p_i) - d(o, p_i)|$. If $d(o, p_i) > d(q, p_i) + r$ or $d(o, p_i) < d(q, p_i) - r$, then $d(q, o) \ge |d(o, p_i) - d(q, p_i)| > r$, 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 *RR*(*r*), we can avoid performing *MR*(*q*, *r*) on wp_i or mp_i. For example, in Fig. 5, the master peer mp_5 does not need to perform *MR*(*q*, *r*) as $M_5 \cap RR(r) = \varnothing$.

To obtain a good estimation of $q.d_k$ (i.e., the k^{th} nearest neighbor distance), we perform a local *MkNN*(*q*, *k*) on the worker peer wp_i with minimum $MIND(MBB(wp_i), \phi(q))$, and use $q.d_k^i$, the distance between *q* and the k^{th} nearest neighbor returned by the local *MkNN*(*q*, *k*) performed by worker peer wp_i as an estimation of $q.d_k$. We consider $q.d_k^i$ as a good overestimation of $q.d_k$. This is because as *q* is located nearest to worker peer wp_i, the value of $MIND(MBB(wp_i), \phi(q))$ reflects the likelihood that *k*NN objects of q are actually located within wp_i. Based on the definition of $q.d_k^i$, we can convert an M*k*NN search into a MR query, as stated in Lemma 2.

Lemma 2. Given a pivot set P, if an object o is an answer object for $MkNN(q, k)$, then $\phi(o)$ is certainly contained in the mapped range region $RR(q, q.d_k^i)$.

Proof. Assume, to the contrary, that there exists an object $o \in MkNN(q, k)$ but $\phi(o) \notin RR(q, q.d_k^i)$. According to the fact that $\phi(o) \notin RR(q, q.d_k^i)$, we have $d(q, o) > q.d_k^i$. Meanwhile, according to the definition of $q.d_k^i$, we have $q.d_k^i \geq q.d_k$, and hence $d(q, o) > q.d_k$, which contradicts with our assumption that $o \in MkNN(q, k)$. Consequently, the proof completes. □

Based on Lemma 2, M*k*NN query processing in AMDS can also be partitioned into two phases, i.e., $q.d_k$ estimation phase and M*k*NN query phase. First, worker peer wp_i with the minimum $MIND$(MBB(wp_i), $\phi(q)$) is selected to perform a local M*k*NN query to obtain an estimation $q.d_k^i$ of $q.d_k$. Then, M*k*NN(q, k) is transformed into a MR $(q, q.d_k^i)$. Note that, k is still needed because at most k objects will be sent back to the M*k*NN query poster to reduce the network communication volumes. For worker peers who receive such M*k*NN missions, they perform local $MR(q, q.d_k^i)$, but at most k nearest objects will be sent back to the mission poster. When all the contributors returned their query results, the *poster* will obtain the global *k*NN objects as the final result.

Algorithm 2 MkNNProcessing

```
Input: a peer $p_i$ who is monitoring the mission publication
 1: loop
 2:   $m$ := receiveMission()/receiveMessage()
 3:   if $m$ is a $DEst$ mission then
 4:     if $p_i$ is a root peer then
 5:       $p_j$:= locateNearestMBB($p_i.MBBList$, $m.content$)
 6:       sendMessage($p_i$, $m$, $IP_j$)
 7:     else if $p_i$ is a master peer then
 8:       if $\phi(q) \subset MBB(p_i)$ or $m.sender = root$ then
 9:         $wp_j$ :=locateNearestMBB($p_i.MBBList$,$m.content$)
10:         sendMessage($p_i$, $m$, $IP_j$)
11:       else
12:         sendMessage($p_i$, $m$, IP($p_i.parent$))
13:     else if $p_i$ is a work peer
14:       $R$ = findKNN($p_i$, $q$, $k$)
15:       $q.d_k^i$:= $d(q,R[k])$
16:       sendMessage($p_i$, ($m.ID$, $DEst$, $q.d_k^i$), $m.IP_w$)
```

We develop a M*k*NN_WP Algorithm to publish a mission when a M*k*NN(q, r) is issued at the worker peer wp_i, with the pseudo-code depicted in Algorithm 1. The algorithm takes M*k*NN(q, k) and the issuer wp_i as an input. If MBB of wp_i contains $\phi(q)$, work peer wp_i is confirmed to be the one with minimum $MIND(MBB(wp_i)$, $\phi(q))$ value. A local M*k*NN search is performed to find the distance $q.d_k^i$ between q and its local k^{th} nearest object, and then we perform a MR query with radius set to $q.d_k^i$ (lines 1–4). Here, MR query searches on work peers whose MBBs are intersected with $RR(q.d_k^i)$ due to Lemma 1, which is simple, and thus, the codes are omitted. On the other hand, if the bounding box of the query issuer does not bound the query point, we need to find the worker peer wp_j with minimum $MIND(MBB(wp_j), \phi(q))$ value, via a mission with *type* = *DEst* (lines 6–10). Once $q.d_k^i$ is located and returned, a metric range query based on q and $q.d_k^i$ is issued (line 11). Among the objects that are located within the search range, the top k objects with minimum distances to q are returned as the global *k*NN objects to complete the process.

We also develop an M*k*NNProcessing algorithm to explain how the estimation of the search radius can be performed by other peers, with its pseudo code listed in Algorithm 2. All the actions are triggered by new missions received, and the actions of different types of peers vary. For M*k*NN query processing, the objective of actions that are triggered by missions/messages with *type* = *DEst* is to find a good estimation of $q.d_k$. As mentioned before, a *DEst* mission is published only when the query object is not located within the MBB of the query issuer mission. The mission first reaches the parent master peer of the query issuer. If the parent master peer has its MBB bounding the query point (the first condition of the IF clause in line 8), it selects a child worker peer wp_j that has the minimum $MIND(MBB(wp_j), \phi(q))$ value via the function locateNearestMBB() and then informs wp_j to performs the estimation via a direct message (lines 9–10). Otherwise, the parent master peer is not able to confirm that it has a shorter *MIND* to $\phi(q)$, as compared with other master peers. It has to ask the root peer for help (lines 11–12). The mission is then propagated to the root peer. The root peer locates the master peer mp_j with minimum *MIND* distance $\phi(q)$ again via function locateNearestMBB() then informs mp_j to perform the estimation via a direct message (lines 4–6). The mission is then propagated to a master peer which might or might not be the parent master peer. Once a master peer receives the *DEst* message from the root peer (the second condition of the IF clause in line 8), it is aware that itself is the nearest master peer to the query point, and it locates the nearest worker peer and informs the worker peer to continue the estimation task (lines 9–10). Now, the mission reaches the destination, the worker peer that is nearest to the query point. The worker peer performs a local *k*NN search, and the distance between q and its local k^{th} NN is returned to the query issuer as an estimation of $q.d_k$. The estimation is ended when a message containing the estimation is sent to the query issuer.

Example 1. We illustrate the M*k*NN query processing using the example shown in Fig. 5, with the corresponding communications depicted in Fig. 1. Suppose that worker peer wp_h raises a M*k*NN query M*k*NN(q, 2) and it invokes *k*NN_WP algorithm. As the query object q locates outside its $MBB(wp_h)$, wp_h publishes a $q.d_k$ estimation mission m to its parent master peer mp_4. Once mp_4 receives m via M*k*NNProcessing Algorithm, it checks whether its MBB bounds q. As q falls inside $MBB(mp_4)$, it locates wp_g, the nearest worker peer among its children, and informs wp_g to continue the estimation via a direct message. Thereafter, worker peer wp_g performs a local M*k*NN query to obtain the result set S_R and $d(q, o_5)$ is returned to wp_h as an estimation of $q.d_k$. In the sequel, wp_h performs a range query with $r = q.d_k^i$. Once the result objects of the range query are received, worker peer can return the top-2 objects $\{o_6, o_7\}$ nearest to q as the result to complete the processing of M*k*NN query.

5.2 Asynchronous Execution of Missions

AMDS adopts the publish/subscribe communication model, which can support asynchronous execution of queries and thus can avoid waiting for communications with other peers. In AMDS, there are three types of characters during the query processing, i.e., the query initiator, the query broker and the query answerer. In particular, a query initiator is a peer which issues a query, a query answerer is a peer which performs the query and return the query answer to the query initiator, and a query broker is a peer that distributes the query to the correct answerers. A query can be divided into four main phases, query raising, query distributing, query processing and result collecting. Each of these four phases is processed by these characters independently, i.e., the query initiator, the query brokers, the query answerers and the query initiator, respectively. It is obvious that the four phases are loosely coupled, no strong relations between these phases exist, which is the premise of asynchronous execution.

Consider the example of asynchronous execution shown in Fig. 6. AMDS consists of two worker peers (i.e., wp_x and wp_y) and one master peer mp. Each of the worker peers issues a query (i.e., q_1 and q_2) that both wp_x and wp_y are related with the query. Although q_1 is finished earlier in Fig. 6(a) than that in Fig. 6(b), 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 increase.

Table 2. Parameter settings

Parameters	Value
Cardinality	250K, 500K, **1M**, 2M, 4M
The number of worker peers	1K, 2K, **4K**, 8K, 16K
k	1, 3, **9**, 27, 81

Table 3. Construction cost of AMDS

Dataset	Network
Title	397376 KB
CoPHIR	2157988 KB
VECTOR (250K)	200954 KB
VECTOR (500K)	398878 KB
VECTOR (1M)	792620 KB
VECTOR (2M)	1587004 KB
VECTOR (4M)	3174662 KB

6 Experimental Evaluation

In this section, we evaluate the effectiveness and efficiency of AMDS and M*k*NN queries via extensive experiments, using both real and synthetic datasets. AMDS and corresponding M*k*NN 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. The similarity between two strings is measured using edit-distance. CoPHIR consists of 1000 K standard MPEG-7 image features extracted

[1] Available at http://www.ncbi.nlm.nih.gov/pubmed.

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

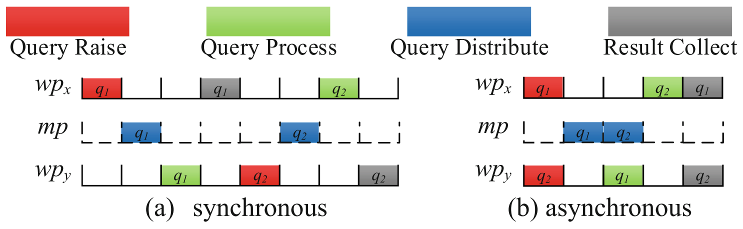


Fig. 6. Comparisons between execution modes

from Flickr[3], where the similarity between two features is measured as the L_2-norm. In addition, synthetic datasets VECTOR are generated with the cardinality varying from 250K to 4M, where L_∞-norm is the distance metric. Every dimension of VECTOR datasets is mapped to [0, 10000]. Each VECTOR dataset has 10 clusters and each cluster follows Gaussian distribution. In this paper, the number of pivots for each dataset is set to 5.

We investigate the performance of AMDS and M*k*NN algorithms under various parameters as summarized in Table 2. In each set of experiments, only one factor varies, whereas the others are fixed to their default values. As discussed in Sect. 4.2, if the number of worker peers is fixed, then the number of master peers will affect the efficiency of AMDS. Hence, in our experiments, the number of master peers is set as 32, 64 and 128 to evaluate the impact of the number of master peers. The main performance metrics include the CPU time and the network communication volume.

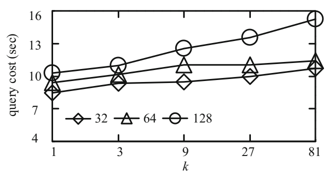
6.1 Construction Cost

The first set of experiments verifies the AMDS construction cost, i.e., the cost of data deployment of AMDS. Here, the network communication volume is used as the performance metric. We collected the construction cost on both real and synthetic datasets, with the results demonstrated in Table 3. The number of worker peers is set to 4 K as default, and the number of master peers is set to 64 as default. The first observation is that the data deployment in AMDS is efficient in terms of the network communication volume. This is because, 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 larger dataset is, the higher construction cost is. This is because the network communication volume depends on the cardinality of dataset.

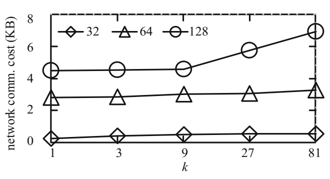
6.2 Evaluation of Metric Similarity Queries

The second set of experiments evaluates the performance of M*k*NN queries using real and synthetic datasets. We study the influence of several parameters, including (i) the value *k*, (ii) the number of worker peers num_{wp}, and (iii) the cardinality of dataset.

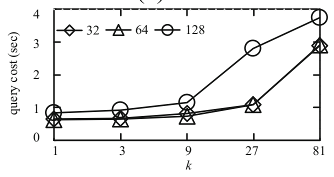
[3] Available at http://www.flicker.com.



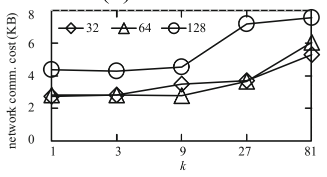
(a) *Title*



(b) *Title*



(c) *CoPHIR*



(d) *CoPHIR*

Fig. 7. Effect of *k*

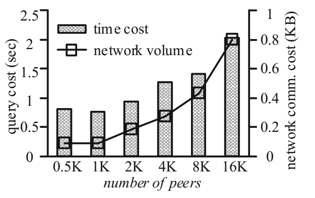


Fig. 8. Effect of num_{wp}

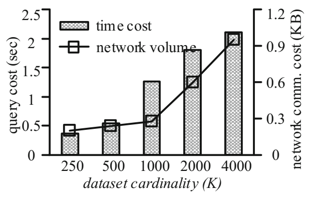


Fig. 9. Effect of cardinality

Effect of k. First, we investigate the performance of M*k*NN queries using real datasets. The CPU time and the network communication volume of M*k*NN queries are shown in Fig. 7 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* increases, resulting in more related master peers and worker peers. Note that, on *CoPHIR*, the CPU time and the network communication volume grows rapidly when *k* exceeds 9 due to the corresponding distance distribution of the dataset.

Effect of Number of Worker Peers. Then, we evaluate the influence of number of worker peers. Figure 8 shows the results under various numbers of worker peers num_{wp} using synthetic datasets. Note that, the number of master peers is set to 64 as default. The first observation is that the query cost first decreases from 0.5K to 1K and then increases from 1K to 16K. This is because, with more worker peers, the objects managed by each worker peer become less, and thus the M*k*NN cost on each worker peer decreases. However, at the same time, more query cost is consumed on the managing of a larger number of peers and communications between peers. In this case, 1K worker peers performs 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. 9. 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.

7 Conclusions

In this paper, we present the Asynchronous Metric Distributed System (AMDS), which aims at dealing with a large scale of M*k*NN queries simultaneously. In the data deployment, AMDS uniformly partitions the data using the pivot-mapping technique to ensure the load balancing. During the M*k*NN query processing, AMDS utilizes the publish/subscribe communication model to support asynchronous processing and achieve the robustness at the same time. In addition, pruning rules are developed with the MBB technique to reduce the query cost. Furthermore, MkNN queries are solved using estimation to avoid high network communication cost. Finally, extensive experiments on real and synthetic datasets verify the efficiency of AMDS construction and M*k*NN 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. This work was supported in part by the 973 Program No. 2015CB352502, the NSFC Grant No. 61522208, and the NSFC-Zhejiang Joint Fund under Grant No. U1609217. Yunjun Gao is the corresponding author of this work.

References

1. Batko, M., Gennaro, C., Zezula, P.: A scalable nearest neighbor search in P2P systems. In: Ng, W.S., Ooi, B.-C., Ouksel, Aris M., Sartori, C. (eds.) DBISP2P 2004. LNCS, vol. 3367, pp. 79–92. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31838-5_6
2. Batko, M., Gennaro, C., Zezula, P.: Similarity grid for searching in metric spaces. In: Türker, C., Agosti, M., Schek, H.-J. (eds.) Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures. LNCS, vol. 3664, pp. 25–44. Springer, Heidelberg (2005). https://doi.org/10.1007/11549819_3
3. Batko, M., Novak, D., Falchi, F., Zezula, P.: Scalability comparison of peer-to-peer similarity search structures. Future Gener. Comput. Syst. **24**(8), 834–848 (2008)
4. Batko, M., Novak, D., Falchi, F., Zezula, P.: On scalability of the similarity search in the world of peers. In: INFOSCALE, p. 20 (2006)
5. Dohnal, V., Sedmidubsky, J., Zezula, P., Novak, D.: Similarity searching: towards bulk-loading peer-to-peer networks. In: SISAP, pp. 87–94 (2008)
6. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Peer-to-peer similarity search in metric spaces. In: VLDB, pp. 986–997 (2007)
7. Traina Jr., C., Filho, R.F.S., Traina, A.J.M., Vieira, M.R., Faloutsos, C.: 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 (2007)
8. Novak, D., Batko, M., Zezula, P.: Large-scale similarity data management with distributed metric index. Inf. Process. Manag. **48**(5), 855–872 (2012)
9. Stoica, I., Morris, R.T., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for internet applications. In: SIGCOMM, pp. 149–160 (2001)

10. Vlachou, A., Doulkeridis, C., Kotidis, Y.: Metric-based similarity search in unstructured peer-to-peer systems. In: Hameurlain, A., Küng, J., Wagner, R. (eds.) Transactions on Large-Scale Data- and Knowledge-Centered Systems V. LNCS, vol. 7100, pp. 28–48. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28148-8_2
11. Ares, L.G., Brisaboa, N.R., Esteller, M.F., Pedreira, O., Places, A.S.: Optimal pivots to minimize the index size for metric access methods. In: SISAP, pp. 74–80 (2009)
12. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. In: SIGMOD, pp. 322–331 (1990)
13. Shen, H.T., Shu, Y., Yu, B.: Efficient semantic-based content search in P2P network. IEEE Trans. Knowl. Data Eng. **16**(7), 813–826 (2004)
14. Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S.: A scalable content-addressable network. In: SIGCOMM, pp. 161–172 (2001)
15. Bawa, M., Condie, T., Ganesan, P.: LSH forest: self-tuning indexes for similarity search. In: WWW, pp. 651–660 (2005)
16. Banaei-Kashani, F., Shahabi, C.: SWAM: a family of access methods for similarity-search in peer-to-peer data networks. In: CIKM, pp. 304–313 (2004)
17. Doulkeridis, C., Nørvåg, K., Vazirgiannis, M.: DESENT: decentralized and distributed semantic overlay generation in P2P networks. IEEE J. Sel. Areas Commun. **25**(1), 25–34 (2007)
18. Jagadish, H.V., Ooi, B.C., Vu, Q.H., Zhang, R., Zhou, A.: VBI-tree: a peer-to-peer framework for supporting multi-dimensional indexing schemes. In: ICDE, p. 34 (2006)
19. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM, pp. 353–366 (2004)
20. Liu, B., Lee, W., Lee, D.L.: Supporting complex multi-dimensional queries in P2P systems. In: ICDCS, pp. 155–164 (2005)
21. Kalnis, P., Ng, W.S., Ooi, B.C., Tan, K.: Answering similarity queries in peer-to-peer networks. Inf. Syst. **31**(1), 57–72 (2006)
22. Ghanem, S.M., Ismail, M.A., Omar, S.G.: VITAL: structured and clustered super-peer network for similarity search. Peer-to-Peer Netw. Appl. **8**(6), 965–991 (2015)
23. Falchi, F., Gennaro, C., Zezula, P.: A content–addressable network for similarity search in metric spaces. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, Aris M. (eds.) DBISP2P 2005-2006. LNCS, vol. 4125, pp. 98–110. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71661-7_9
24. Novak, D., Zezula, P.: M-Chord: a scalable distributed similarity search structure. In: INFOSCALE, p. 19 (2006)
25. Jagadish, H.V., Ooi, B.C., Tan, K., Yu, C., Zhang, R.: iDistance: an adaptive B^+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. **30**(2), 364–397 (2005)
26. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquin, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (2001)
27. Batko, M., Novak, D., Zezula, P.: MESSIF: metric similarity search implementation framework. In: Thanos, C., Borri, F., Candela, L. (eds.) DELOS 2007. LNCS, vol. 4877, pp. 1–10. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-77088-6_1
28. Mühl, G., Fiege, L., Pietzuch, P.R.: Distributed Event-Based Systems. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-32653-7
29. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems - Concepts and Designs. International Computer Science Series, 3rd edn. Addison-Wesley-Longman, Boston (2002)

30. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognit. Lett. **24**(14), 2357–2366 (2003)
31. Yu, C., Ooi, B.C., Tan, K., Jagadish, H.V.: Indexing the distance: an efficient method to KNN processing. In: VLDB, pp. 421–430 (2001)
32. Chen, L., Gao, Y., Li, X., Jensen, C.S., Chen, G.: Efficient metric indexing for similarity search. In: ICDE, pp. 591–602 (2015)