Hindawi Publishing Corporation
Mobile Information Systems
Volume 2015, Article ID 657431, 20 pages
http://dx.doi.org/10.1155/2015/657431

*Research Article*

# Efficient Multidimensional Top-$k$ Query Processing in Wireless Multihop Networks

**Daichi Amagata,[1] Yuya Sasaki,[2] Takahiro Hara,[1] and Shojiro Nishio[1]**

[1] *Graduate School of Information Science and Technology, Osaka University, Osaka 565-0871, Japan*
[2] *Institute of Innovation for Future Society, Nagoya University, Aichi 464-8601, Japan*

Correspondence should be addressed to Daichi Amagata; amagata.daichi@ist.osaka-u.ac.jp

Received 29 August 2014; Accepted 1 September 2014

Academic Editor: David Taniar

Top-$k$ queries, which retrieve the $k$ most preferable data objects, have been receiving much attention. An emerging challenge is to support efficient top-$k$ query processing in a wireless distributed network. In this study, we investigated how to process multidimensional top-$k$ queries efficiently in a wireless multihop network. A major challenge for multidimensional top-$k$ queries is that answers for different users are typically different, because each user has a unique preference and search range. Meanwhile, it is desirable for wireless networks to reduce unnecessary traffic even if users issue top-$k$ queries with their own unique preferences. Therefore, we address the above problem and propose a top-$k$ query processing method in wireless multihop networks, called *ClusTo*. ClusTo performs a novel clustering scheme for multidimensional top-$k$ query processing and routes queries based on the cluster while guaranteeing the user's specified search range. Moreover, ClusTo takes a dynamic threshold approach to suppress unnecessary query transmissions to nodes which do not contribute to top-$k$ data retrieval. Extensive experiments on both real and synthetic data have demonstrated that ClusTo outperforms existing methods in terms of traffic and delay.

## 1. Introduction

Recent remarkable innovations in engineering have been bringing us to variety of distributed networks and wireless communication mediums. Information retrieval in wireless distributed networks presents an important challenge of supporting efficient query processing [1]. Ranking queries, for example, top-$k$ queries, enhance efficiency of data retrieval and can help suppressing huge result sets and provide users with only their required result. Top-$k$ queries retrieve only the $k$ data objects that best match the user's specified query condition (scoring function) [2–4]. A wide range of applications, such as multimedia retrieval in P2P networks [5–9], environment monitoring in wireless sensor networks [10–12], and, in ad hoc networks, military situations and rescue operation in disaster sites [13–15], realize considerable benefits from top-$k$ query processing.

As introduced above, the systems operated in the applications are becoming increasingly distributed. Thus, many researchers nowadays engage in studying efficient query processing in distributed environments. The emerging and important challenge which we tackle in this paper is data retrieval in wireless distributed networks with no infrastructure support, and we focus on multidimensional top-$k$ query processing in such environments, where users can specify a monotone scoring function and search range as a query condition. One of the representative examples is data retrieval in a wireless sensor network. A user wants to collect top-3 sensor readings existing around a sensor node (e.g., within 100 meters) to investigate environment details. The sensor readings consist of some attributes such as temperature and humidity. If the user cares about temperature more than humidity, the users specify the scoring function which weights temperature higher than humidity and the search range as 100 meters. Another example is a cooperative or group working in a site where there is no infrastructure, and each of the group members is assigned a small region for investigation. In such application, users may retrieve data objects by utilizing wireless communication mediums. For example, a user wants to obtain top-2 data existing around the user, where data objects are represented by some attributes, for example, 2-dimensional data. To obtain his/her preferable data, s/he may specify the scoring function as "60% priority to attribute $A$ and 40% to attribute $B$" and the search range as "200 meters" (we describe the definition in Section 2), and, through wireless peer-to-peer communication, s/he can receive the top-2 data objects among the set of data objects held by the user and others in the group, without any infrastructure supports.

Due to the wireless communication, packet losses and wireless nodes' energy consumption become severe problems as traffic increases. It is therefore important to suppress unnecessary traffic by reducing the number of data replies not included in the exact result (we call this the top-$k$ result) and the number of nodes to which a query is transmitted. Ideally, top-$k$ query processing by only nodes that contribute to retrieving the top-$k$ result is preferable. The top-$k$ result, on the other hand, varies according to users' specified query conditions and $k$. Two simple approaches may be applied to processing a top-$k$ query. One is that the query originator floods a query message over all the nodes in the search range and then collects the local top-$k$ data objects from all query receiving nodes. The problem here is that the query message is transmitted to many unnecessary nodes which do not hold data objects included in the top-$k$ result. The other is to construct efficient query transmission routes with respect to each query condition. However, if the query conditions are different for different users (i.e., the query conditions are diverse), it is necessary to construct and maintain too many query transmission routes, resulting in dramatic increase of overhead. On the other hand, if every node distributes its own data objects which might be included in the top-$k$ results to all nodes in the network, they can avoid distributed query processing. The drawbacks of this approach are that too many data objects are distributed, and too much traffic is involved when data update occurs. Such massive data distribution actually generates too much traffic and then misses necessary data objects because of packet losses. As seen above, streamlining multidimensional top-$k$ query processing is difficult, but processing a top-$k$ query with small traffic and delay is nonetheless important.

In this paper, we propose ClusTo (Cluster and $K$-skyband-based multidimensional top-$k$ query processing), which processes a top-$k$ query with small traffic and delay even if the query conditions are diverse. ClusTo employs a new and novel clustering framework with skyline [16], which makes nodes holding data objects which are expected to be included in many top-$k$ results become cluster-heads. This clustering framework provides multidimensional top-$k$ query processing without transmitting queries to all nodes. Moreover, in the procedure in clustering, each node acquires the information on its neighboring nodes' $K$-skyband [17]. $K$-skyband helps to construct a filter, which can prune replying unnecessary data objects and query transmissions to unnecessary nodes, and the filter is attached to a query message. ClusTo routes queries to cluster-heads within user's specified search range with small traffic, and then the cluster-heads transmit the query to nodes, based on a threshold derived from the filter that contribute to retrieving the top-$k$ result. As a consequence, ClusTo can retrieve the top-$k$ result from a small number of nodes in the search range. The contributions of this paper are as follows.

(i) We propose a novel and scalable top-$k$ processing method, ClusTo, which employs a new clustering framework with skyline and a query routing scheme. This method performs multidimensional top-$k$ query processing with small traffic and delay even if the query conditions are diverse.

(ii) The proposed approach is adaptable to wireless network environments such as unstructured P2P networks, sensor networks, and ad hoc networks.

(iii) We evaluated our proposed method through extensive experiments on both real and synthetic data, and the results show that ClusTo functions well in terms of traffic and delay.

The reminder of this paper is organized as follows. Section 2 provides the preliminaries to present ClusTo. Section 3 reviews the related work. In Section 4, we describe the detail of ClusTo. We show the results of our simulation experiments in Section 5. Finally, in Section 6, we conclude our paper.

## 2. Preliminaries

In this section, we explain a data model, a network model, and definitions of top-$k$ queries and some other concepts.

*2.1. Data Model.* We assume that each data object has unique identifier, $i$, and $m$ metadata $v_i[j]$ ($1 \leq j \leq m$), which represents the features of data objects, and data objects are not replicated. Each metadata value is calculated by some kind of numerical scoring function and follows Euclidean space. Without loss of generality, we assume that a smaller value is preferable. We further assume that the value of each metadata is normalized between $[0, value_{max}]$. In this paper, we define the number of dimensions of data objects as the number of metadata, $m$. A data object consists of not only $m$ metadata, but also some other contents such as images and text information, for example, documents. Thus, the size of data object is basically larger than the summation of the size of metadata.

*2.2. Network Model.* A wireless network consists of $n$ nodes, which are assigned unique identifier, such as $\{N_1, N_2, \dots, N_n\}$. Each node communicates by identical wireless communication standard with radio communication range of $r$ [m], so node $N_i$ can communicate with node $N_j$ if the distance between them is not more than $r$. We assume that each node can recognize its accurate location information via GPS. In addition, we assume a static network; that is, the network topology does not change and node failure does not occur. Although there may be some applications that users holding a portable device (node) move, we assume that they move in a small range, which does not affect the network topology. For example, in a wireless network for cooperative data retrieval applications, each user does not move selfishly

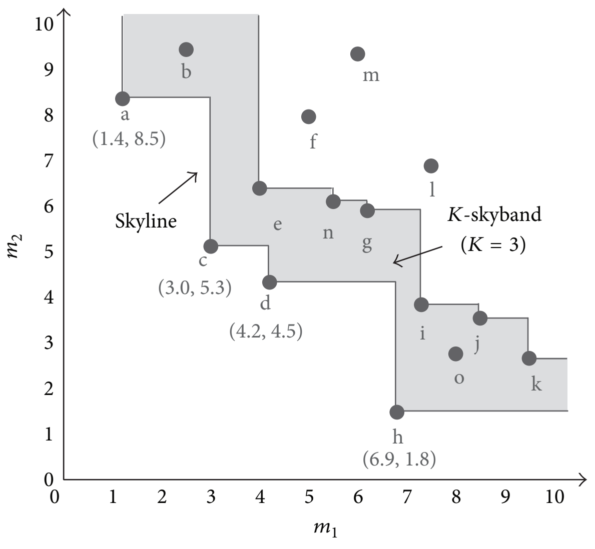


Figure 1: An example of skyline and $K$-skyband ($K = 3$).

and possibly moves only in a limited or small area, which is assigned to the user.

*2.3. Definition.* Top-$k$ queries, in this paper, give a score to a data object $o_i$ from its metadata with increasingly monotone scoring function, $f : \mathbb{R}^d \rightarrow \mathbb{R}$. Specifically, the score of $o_i$ is calculated as follows:

$$score(o_i) = f(o_i) = \sum_{j=1}^{m} w_j \times v_i[j], \tag{1}$$

where $w_j$ ($1 \leq j \leq m$) is a weighting factor (value) for each dimension $m_j$ and nonnegative value ($\sum_{j=1}^{m} w_j = 1, w_j \geq 0$). These weighting factors indicate user preference and influence the ranking of data objects. For example, Figure 1 shows a 2-dimensional dataset and depicts data objects, `a` to `o`. If a high weight is assigned to $m_1$, data object `a` becomes the top-1 data object, while if a high weight is assigned to $m_2$, data object `h` becomes the top-1 data object. The scoring function, $f$, is a weighting sum (linear) function and has a following property:

$$\forall j \in [1, m], \quad v_i[j] \leq v_{i'}[j] \Longrightarrow f(o_i) \leq f(o_{i'}). \tag{2}$$

Furthermore, this top-$k$ query allows users to input a search range, $d$ [m]. Users can retrieve $k$ data objects within $d$ [m] of their locations. Thus, data objects, which exist in the search range, with the $k$ best scores are included in the top-$k$ query result. The formal definition of top-$k$ result is provided as follows.

*Definition 1* (top-$k$ result). We define the top-$k$ result as the data objects with the $k$-lowest scores held by nodes within $d$ [m] of a node (user) generating a query.

There is a maximum $k$ ($K$) which is specified by the system. We assume that $K$ is not large, for example, 15. Due to the bandwidth limitation in wireless networks, communication failures and packet losses frequently occur if $K$ is large; that is, traffic is large, resulting in a loss of necessary data objects. In addition, applications such as data retrieval [18] and environment monitoring [10], for example, can deal with basically only a small number of data objects. (Even web search, e.g., Google, typically displays only ten web sites by default.) This assumption is therefore practical in terms of both users and the system; however our proposed method can be adapted to cases where $K$ is large.

In the past, some important concepts related to top-$k$ queries have been developed. ClusTo exploits them, so we define such concepts below.

*Definition 2* (dominant). Consider data objects $o_i$ and $o_{i'}$. $o_i$ is said to dominate $o_{i'}$ if they satisfy the following condition:

$$\begin{aligned}(\forall j \in [1, m], v_i[j] \leq v_{i'}[j]) \\ \land (\exists j \in [1, m], v_i[j] < v_{i'}[j]).\end{aligned} \tag{3}$$

In Figure 1, data object `c` dominates `e`, `f`, `g`, `l`, `m`, and `n`.

This dominance relationship provides us with an important dataset, that is, skyline [16]. The definition of skyline is shown as follows.

*Definition 3* (skyline). Skyline is the set of data objects which are not dominated by any other data objects.

In Figure 1, skyline contains data objects `a`, `c`, `d`, and `h`.

An interesting observation from skyline shows a key to develop a novel approach for query routing, and we give a lemma explaining the observation.

**Lemma 4.** *The top-1 data object for any increasingly monotone function belongs to the skyline.*

$K$-skyband enhances top-$k$ query processing more, and we define this dataset below and then give an important lemma.

*Definition 5* ($K$-skyband). $K$-skyband is the set of data objects which are dominated by at most $K - 1$ other data objects.

**Lemma 6.** *The top-k data objects for any increasingly monotone function belong to the K-skyband.*

*Proof.* Assume that data object $o_i$ does not belong to $K$-skyband. There exists at least $K$ other data objects that dominate it. According to (3), there exists at least $K$ data objects whose score is less than $o_i$. Consequently, $o_i$ cannot be top-$K$ data object. Skyline is the special case where $K = 1$. Therefore, since Lemma 6 is true, Lemma 4 is also true. □

## 3. Related Work

Many studies have proposed top-$k$ query processing methods in distributed environments as well as centralized fashion [2–4, 19]. Here, we review some representative methods.

In [7], the authors proposed an algorithm called TPUT, which aims to prune unnecessary data objects. TPUT was improved by KLEE [8]. These methods need multiple phases, so the delay increases if adapted to wireless multihop networks. Moreover, data objects are vertically distributed, which is different from our assumption.

In [5, 6, 9], data objects are horizontally distributed to each node, which is the same assumption in this paper. Although the authors in [6] proposed a method that minimizes data replies, the method involves all nodes, which is inefficient for wireless networks. The method proposed in [5] can avoid query flooding by selecting nodes to access based on scores of data objects. However, this method assumes that each node can communicate with any other nodes, which is different from our assumption. In [9], the authors proposed SPEERTO which utilizes a super-peer network. This method performs top-$k$ query processing through super-peers. Even though this method is efficient for highly distributed environments to process top-$k$ queries, this method is hard to adapt to wireless networks with limited communication range. In SPEERTO, it is necessary to communicate with some other super-peers to process top-$k$ queries, so if adapting this method to a wireless multihop network, many or most of nodes have to perform the role of super-peer due to the limited communication range. This is obviously inefficient since this approach involves many unnecessary nodes to retrieve top-$k$ data, and actually it is impossible to adapt such a special overlay network to wireless networks, without no infrastructure support. Therefore, an efficient multidimensional top-$k$ query processing technique for the environments that we assume is required, and we propose an effective clustering framework specialized for multidimensional top-$k$ query processing, which exploits metadata values of data objects held by nodes. This clustering framework does not need special overlay such as the super-peer, meaning that no infrastructure support is required.

Many papers [10–12, 20–22] deal with efficient processing of top-$k$ queries in wireless sensor networks with the aim of prolonging network life time. An in-network aggregation technique has been proposed in [11]. Every node sends its own top-$k$ data objects in each epoch in this method. Although the method proposed in [23] processes a top-$k$ query as in [11], this method replies to not only parent node but also its brother node to provide them with more information on the scores of data objects, which can prune more unnecessary data objects. In [24], the authors proposed a method that suppresses energy consumption which sacrifices the accuracy of the query result. The method proposed in [22] optimizes top-$k$ queries by employing a sampling-based approximation. However, this approximation technique does not guarantee perfect accuracy of the query result. The authors in [12, 20, 21] proposed methods which utilize a filtering approach to prune unnecessary data objects. A sink node in the network constructs filters from received data objects and distributes the filters to each sensor node. When updating the sensor readings, each sensor node judges whether or not to send the reading to the sink based on the filter. The method which retrieves the top-$k$ result fast (within a certain upper bound time) has been proposed in [25]. Unfortunately, all the above methods deal with *only 1-dimensional* data objects. Enhanced scheme proposed in [10] is the only method that deals with (continuous) multidimensional top-$k$ queries. This method utilizes the *dominant graph* [26] which maintains the dominance relation of data objects. The sink node constructs a filter, utilizing this dominant graph, to prune data objects which are never included in any top-$k$ results. However, this method considers all possible query conditions that are not inserted in the system, which results in redundant increases of traffic. Note that, in wireless sensor networks assumed in the above existing studies, only the sink node can be a query originator, but in this paper, we assume that all nodes can be query originators, that is, different assumption of such wireless sensor networks. ClusTo performs efficient multidimensional top-$k$ query processing even in a case where all nodes generate queries.

Top-$k$ query processing methods have also been proposed for mobile ad hoc networks [13–15, 27]. The method proposed in [27] assumes that the scores of data objects follow some kind of distribution such as Gaussian distribution. However, because such scores are dependent of query conditions in multidimensional top-$k$ queries, this assumption is impractical. To suppress unnecessary data replies, the method proposed in [14] estimates the $k$th score of data objects in the network by attaching *standard scores* to query messages. The aim of the method proposed in [15] is to prune more data objects which are not included in the top-$k$ result. This method adapts 2-phase query processing. In the first phase, a query originator collects scores of data objects and sets the $k$th lowest score as a threshold. Then, in the second phase, the query originator floods a query message attached with the threshold, and nodes receiving the query reply data objects with scores not larger than the threshold. The common drawback of these methods is that all nodes are involved in top-$k$ query processing due to query flooding. This drawback increases unnecessary traffic and packet losses, resulting in a lack of scalability. Although the method proposed in [13] performs efficient query routing to prune unnecessary query paths to retrieve the top-$k$ result, this method generates excessive traffic if the query conditions are diverse because query routes are constructed according to each query condition. Thus, this method is unsuitable for multidimensional top-$k$ queries. In contrast, ClusTo can perform efficient query processing without transmitting queries to all nodes within the search range even if the query conditions are diverse. It is not necessary, furthermore, to construct query routes with respect to query conditions in ClusTo; that is, clustering is performed only once. This suppresses overhead to construct query routes.

## 4. Proposed Method

In this section, we first introduce the overview of ClusTo. Then we describe the detail of ClusTo, in the order corresponding to cluster formation (Section 4.2), filter construction (Section 4.3), and top-$k$ query processing (Section 4.4).

*4.1. Overview.* Our objective is to suppress unnecessary traffic and delay, and to this end, ClusTo aims to process top-$k$

queries by only nodes that contribute to retrieving the top-$k$ result as possible. Intuitively, it is effective to construct query routes as this can avoid the query flooding in processing of top-$k$ queries. As mentioned in Section 1, however, constructing query routes for each query condition is inefficient. The key to solving this problem lies in constructing efficient query routes *for any query conditions*. In addition, since $K$-skyband is the set of data objects included in the top-$k$ ($k \leq K$) results, acquiring the information on neighboring nodes' $K$-skyband is an effective mean for reducing unnecessary traffic. This is because nodes can prune unnecessary query transmissions to neighboring nodes which do not hold data objects included in the top-$k$ result by acquiring information on their $K$-skyband. Based on these ideas (constructing efficient query routes and utilizing neighboring nodes' $K$-skyband), we have developed a novel clustering framework. ClusTo first performs a new 1-hop clustering framework exploiting skyline as preprocessing. This framework makes nodes holding data objects with high probability of being included in many top-$k$ results become cluster-heads. By doing so, unnecessary query transmissions can be significantly suppressed because cluster-heads hold many data objects with lower scores. In addition, each node sends metadata of data objects included in its $K$-skyband to its neighboring nodes during the clustering procedure (detail in Section 4.2).

In this clustering framework, each node can acquire the information on its neighboring nodes' $K$-skyband, and each query receiving node utilizes this information to prune unnecessary data objects by constructing a filter (detail in Section 4.3). Query messages are transmitted over the user's specified search range via cluster-heads while establishing parent-child relationships, that is, data forwarding tree. If the query messages are transmitted to the border of the search range, each query receiving node replies to its parent with data objects with scores not larger than the threshold calculated from the filter. When cluster-heads have received reply messages from all their children, they judge whether or not to send a query message to nodes included in their cluster. Cluster-heads send a query message to nodes holding data objects with scores not larger than the threshold. This approach guarantees query transmission to nodes holding data objects included in the top-$k$ result. If there does not exist such nodes or received reply messages from all queried nodes, cluster-heads send a reply message to their parent.

In this way, ClusTo performs efficient multidimensional top-$k$ query processing exploiting effective clustering and $K$-skyband. Next, we describe the detail of each procedure in ClusTo.

*4.2. Cluster Formation.* In this subsection, we explain the procedure of 1-hop clustering framework which realizes multidimensional top-$k$ query processing by less number of nodes for any query conditions.

Flooding is a simple approach to transmit queries to nodes holding data objects included in the top-$k$ result. However, this approach generates redundant traffic because of unnecessary transmissions to nodes that do not contribute to top-$k$ data retrieval. ClusTo overcomes this problem by query routing based on the clustering approach. In cluster-based routing, queries are transmitted between cluster-heads via gateway-nodes which have links to nodes in other clusters. Note that if cluster-heads hold data objects which dominate many data objects held by nodes in their clusters, it is rarely necessary to transmit queries to these nodes in the clusters. Following this idea, ClusTo designates as cluster-heads nodes holding data objects which are expected to be included in many top-$k$ results to realize top-$k$ query processing by less number of nodes. Specifically, each node calculates the skyline for its own data objects (local skyline) and sets a timer to become a cluster-head based on the following equations:

$$Timer = \frac{\alpha \cdot SkyAvgDist + (1-\alpha) \cdot SkyMinDist}{\beta}, \quad (4)$$

$$SkyAvgDist = \frac{\sum_{i=1}^{|skyline|}\left(\sqrt{\sum_{j=1}^{m} v_i[j]^2}\right)}{|skyline|}, \quad (5)$$

$$SkyMinDist = \min_{1 \leq i \leq |skyline|}\left(\sqrt{\sum_{j=1}^{m} v_i[j]^2}\right). \quad (6)$$

$\alpha \in [0,1]$ and $\beta(>0)$ in (4) are system parameters and $\beta$ is used to avoid vain larger value of timer. $|skyline|$ in (5) and (6) represents the number of data objects in the local skyline. *SkyAvgDist* and *SkyMinDist* represent the average distance and minimum distance between local skyline points and the origin point $(0, 0, \ldots, 0)$, respectively. For example, assume that a node holds data objects shown in Figure 1. In this case, the *SkyAvgDist* is $(\sqrt{(1.4^2 + 8.5^2)} + \sqrt{(3^2 + 5.3^2)} + \sqrt{(4.2^2 + 4.5^2)} + \sqrt{(6.9^2 + 1.8^2)})/4 = 7.0$. The *SkyMinDist* is $\sqrt{(3^2 + 5.3^2)} = 6.1$. It is intuitively clear that data objects held by a node have a greater probability of being included in many top-$k$ results if the values of *SkyAvgDist* and *SkyMinDist* are smaller.

Clustering in ClusTo is performed according to Algorithms 1 and 2. The smaller the value of the timer set by a given node is, the shorter the timer will expire, and a node becomes a cluster-head (CH) if its timer expires. A cluster-head broadcasts a cluster formation message ($cf$), which is represented as $cf = \langle$CH_id, sender_id, sender_position, $KSKY\rangle$. The first element, CH_id, is the identifier of a cluster-head. The second element, sender_id, is the identifier of the sender node of this message. If a cluster-head sends this message, CH_id is equal to sender_id. The third element, sender_position, is the location information on the sender node of this message, and the forth element $KSKY$ includes the identifiers and *metadata* of data objects which are $K$-skyband in the set of data objects held by the sender node. Nodes receiving a $cf$ from a cluster-head cancel their timers and then execute the following procedures. (i) They construct their routing table (lines 8 and 13 in Algorithm 1). Each entry in a routing table consists of the identifier of a cluster-head, location information on the cluster-head, and the identifier of a neighboring node for the transmission of queries (next_node) as shown in Figure 2. (ii) They record the neighboring nodes' location information and $KSKY$ (line 14 in Algorithm 1). We emphasize that $KSKY$ does not include data objects, but only metadata. Because

```
(1) /* Expire the timer of node $N_p$ */
(2) $cf \leftarrow \langle p, p, N_p.\text{position}, KSKY_p \rangle$
(3) Broadcast $cf$
(4) /* Node $N_q$ receives a $cf$ from node $N_{p'}$ */
(5) if $cf$.CH_id $\neq q$ then
(6)     if CH_id = sender_id then
(7)         Cancel its own timer
(8)         RoutingTable.entry $\leftarrow$ RoutingTable.entry $\cup \{p', N_{p'}.\text{position}, p'\}$
(9)         if $N_q$ received $cf$ for the first time then
(10)            $cf \leftarrow \langle p', q, N_q.\text{position}, KSKY_q \rangle$
(11)            Broadcast $cf$
(12)    else
(13)        RoutingTable.entry $\leftarrow$ RoutingTable.entry $\cup \{cf.\text{CH\_id}, \emptyset, p'\}$
(14) $Neighbors_q \leftarrow Neighbors_q \cup \{p', N_{p'}.\text{position}, KSKY_{p'}\}$
```

Algorithm 1: Handling cluster formation message.

```
(1) /* Procedure of non-cluster-head, $N_q$ */
(2) $CHL \leftarrow \{\forall \{p, N_p.\text{position}\} \in N_q.\text{RoutingTable}\}$
(3) $mn \leftarrow \langle q, CHL \rangle$
(4) Send $mn$ to the nearest cluster-head from $N_q$
(5) /* Procedure of cluster head $N_p$ which receives $mn$ from $N_q$ */
(6) $CM \leftarrow CM \cup \{q\}$
(7) for $\forall mn.CHL$.entry do
(8)     if $\exists i, j$, RoutingTable.entry$_i$.CH = $mn.CHL$.entry$_j$.CH then
(9)         RoutingTable.entry $_i$.CH_position $\leftarrow$ $mn.CHL$.entry $_j$.position
(10)        if $Dist$(RoutingTable.entry $_i$.CH, $q$) < $Dist$(RoutingTable.entry $_i$.CH, RoutingTable.entry $_i$.next_node) then
(11)            RoutingTable.entry $_i$.next_node $\leftarrow q$
(12)    else
(13)        RoutingTable.entry $\leftarrow$ RoutingTable.entry $\cup \{mn.CHL.\text{entry}, q\}$
```

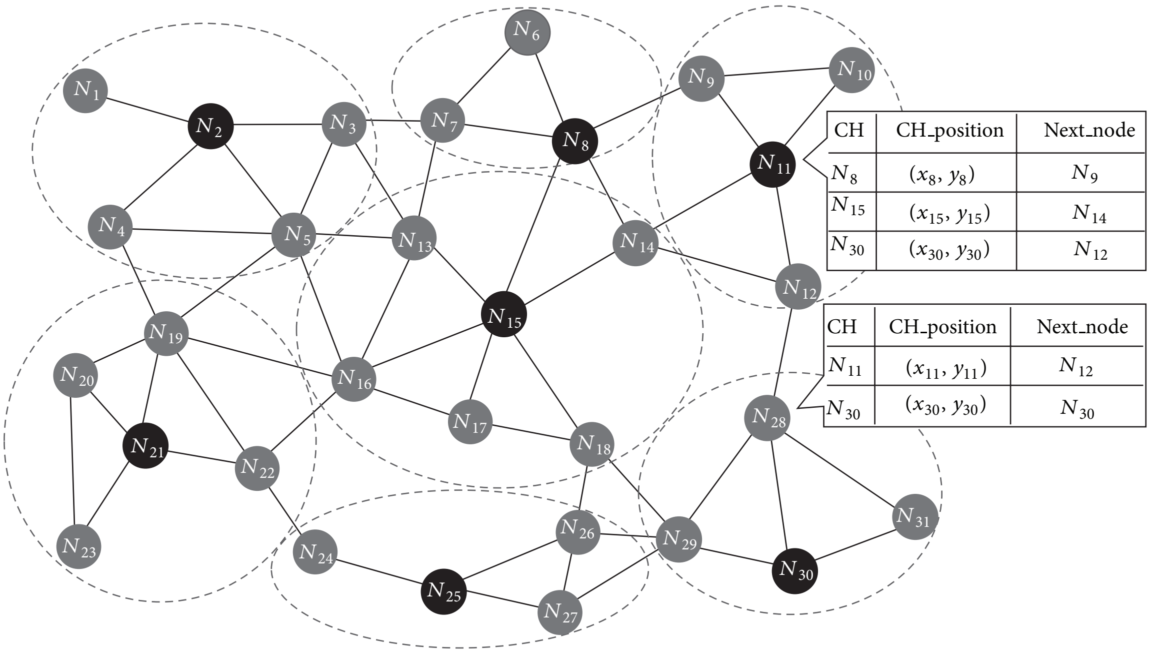
Algorithm 2: Handling member notice message.

metadata is needed to calculate the score of data object, it is only necessary for nodes to obtain metadata. Distributing $K$-skyband to all nodes generates too much overhead because of the reason mentioned in Section 1. Distributing *KSKY* to only neighboring nodes generates small overhead, and utilizing the *KSKY*, nodes can construct effective filter for any query conditions (Section 4.3). (iii) They broadcast the *cf* if they have received it for the first time from a cluster-head (lines 9–11 in Algorithm 1).

After a certain time has elapsed, noncluster-head nodes send a member notice message (*mn*) to their nearest cluster-head. The *mn* is represented as $mn = \langle \text{sender\_id}, CHL \rangle$. The first element is the identifier of the sender node of this message. The second element is a cluster-head list consisting of a pair of cluster-head identifiers and their location information. This procedure provides cluster-heads with the information on cluster-members (*CM*) included in their cluster. In addition, each cluster-head can update its routing table (i.e., know its neighboring cluster-heads and next_nodes) by receiving this message. To reduce query transmission hop counts, each cluster-head designates the nearest nodes from its neighboring cluster-heads (among the nodes from which it has received an *mn*) as next_nodes (lines 10-11 in Algorithm 2), because such nodes tend to have a link to their corresponding cluster-heads.

Following the above procedure, ClusTo forms effective clusters for efficient top-$k$ query processing and limits the number of nodes to which queries are transmitted. During the clustering process, nodes obtain their neighboring nodes' location information and the $K$-skyband information. Even in the case of data updates, nodes holding updated data objects simply have to recalculate their $K$-skyband and broadcast necessary information, that is, updated metadata in *KSKY* and metadata of data objects newly included in *KSKY*. Nodes receiving this information update corresponding neighboring node's *KSKY*.

*4.3. Filter Construction.* ClusTo performs query routing based on a routing table and transmits queries over the user's specified search range. After that, to guarantee query transmission to nodes holding data objects included in the top-$k$ result, each cluster-head transmits a query message to its cluster-members which have yet to receive the query. Notice that it is redundant to transmit queries to nodes which do not contribute to retrieving the top-$k$ result (noncontributor). It is therefore effective to estimate the $k$th lowest



| CH | CH_position | Next_node |
|---|---|---|
| $N_8$ | $(x_8, y_8)$ | $N_9$ |
| $N_{15}$ | $(x_{15}, y_{15})$ | $N_{14}$ |
| $N_{30}$ | $(x_{30}, y_{30})$ | $N_{12}$ |

| CH | CH_position | Next_node |
|---|---|---|
| $N_{11}$ | $(x_{11}, y_{11})$ | $N_{12}$ |
| $N_{30}$ | $(x_{30}, y_{30})$ | $N_{30}$ |

FIGURE 2: An example of cluster and routing table.

score of data objects (threshold) in the search range. This suppresses unnecessary transmissions to noncontributors. An approach whereby nodes attach to the query message the $k$-lowest scores and corresponding identifiers of data object as a filter, and then query receiving nodes estimate the threshold based on the received filter, their own data objects, and their neighbors' *KSKY*, is intuitively applicable. If $k$ is large, however, filter transmission overhead becomes large. Therefore, to suppress such overhead, ClusTo employs not $k$, but three specific scores of data objects; that is, we suppress the overhead for attaching a filter by using a fixed number of scores (and their corresponding data object identifiers). Specifically, the filter consists of the lowest score (lower bound), the $(\lfloor k/2 \rfloor + 1)$th score (median), the $k$th score (upper bound), and their corresponding data object identifiers (we define these pairs of scores and identifiers as *lb*, *me*, and *ub*, resp.). Each node ranks its own data objects and neighbors' *KSKY* by calculating their scores. To ensure that the threshold approximates the exact threshold, query receiving nodes update the filter based on the received filter, their own data objects, and their neighbors' *KSKY*.

Each node basically constructs the filter based on the information on its own data objects, neighbors' *KSKY*, and the received filter. Because the filter consists of three score and identifier pairs, if $k > 3$, a query receiving node might not acquire the complete information on the $k$-lowest scores and their corresponding data object identifiers among the set of data objects held by nodes on the query route from a query originator to the query sender node, which has transmitted the query to the query receiving node, and their neighboring nodes (we define this dataset as the *upstream dataset*). In this case, the node needs to estimate the missing information on scores which the node could not acquire by some kind of way. However, if we roughly estimate the $k$th score, nodes may be provided with a smaller threshold than the exact threshold, which results in query transmission failure to nodes holding data objects included in the top-$k$ result. On the other hand, if we estimate safely, that is, estimate the scores at a large value to guarantee access to all nodes holding data objects included in the top-$k$ result, unnecessary query transmissions and data object replies increase. Thus, it is important not to set the threshold as large unnecessarily with guaranteeing the query transmission to nodes which hold data objects included in the top-$k$ result.

Algorithm 3 illustrates the filter construction algorithm. In ClusTo, each query receiving node estimates the information on the $k$-lowest scores and their corresponding identifiers in the upstream dataset based on the information on its own data objects and its neighbors' *KSKY*. Specifically, query receiving nodes initially calculate the scores of data objects held by nodes in common communication range with their query sender nodes. From these scores and the received filter, they estimate the $k$-lowest scores and their corresponding data object identifiers in the upstream dataset (lines 5–12). In the case where the number of calculated scores, $k'$, whose values are between $[lb.\text{score}, ub.\text{score}]$ is less than the number of data objects (there exist $(\lfloor k/2 \rfloor - 1)$ data objects between $(lb, me)$ and $(k - \lfloor k/2 \rfloor - 2)$ data objects between $(me, ub)$) existing between $(lb, ub)$, that is, $k' < (\lfloor k/2 \rfloor - 1) + (k - \lfloor k/2 \rfloor - 2)$ accurately, such nodes estimate the missing data object information at *me* in $[lb, me]$ and *ub* in $(me, ub]$ (lines 14–16 and 18–20). Supplementing the score information which cannot be acquired with respective maximum scores in $[lb, me]$ and $(me, ub]$ does not estimate the $k$th score of data object in the upstream dataset at less value than the exact value; that is, no false negative

```
    Input: $lb, me, ub$
    Output: $lb', me', ub'$
(1) /* Node, $N_p$, constructs a filter from a query sent by $N_q$ */
(2) $local\ top\text{-}k\ data \leftarrow \emptyset$
(3) $lm\_data \leftarrow mu\_data \leftarrow \emptyset$
(4) $lm\_data_{com} \leftarrow mu\_data_{com} \leftarrow \emptyset$
(5) for $\forall N_{q'} \in Neighbors_p$ do
(6)     if $N_{q'}$ is within search range then
(7)         if $Dist(N_q, N_{q'}) \le r$ then
(8)             $lm\_data_{com} \leftarrow lm\_data_{com} \cup \{\forall \{score(o_j), j\} \mid$
(9)             $o_j \in N_{q'}$'s $K$-skyband, $score(o_j) \in [lb.score, me.score]\}$
(10)            $mu\_data_{com} \leftarrow mu\_data_{com} \cup \{\forall \{score(o_j), j\} \mid$
(11)            $o_j \in N_{q'}$'s $K$-skyband, $score(o_j) \in (me.score, ub.score]\}$
(12) Sort $lm\_data_{com}$ and $mu\_data_{com}$ by ascending order
(13) $lm\_data \leftarrow lb \cup lm\_data_{com}$
(14) if $|lm\_data| < \lfloor k/2 \rfloor + 1$ then
(15)     for $i = 0$ to $\lfloor k/2 \rfloor + 1 - |lm\_data|$ do
(16)         $lm\_data \leftarrow lm\_data \cup \{me.score, -(i+1)\}$
(17) $mu\_data \leftarrow mu\_data_{com}$
(18) if $|mu\_data| < k - \lfloor k/2 \rfloor - 1$ then
(19)     for $i = 0$ to $k - \lfloor k/2 \rfloor - 1 - |mu\_data|$ do
(20)         $mu\_data \leftarrow mu\_data \cup \{ub.score, -(i+k)\}$
(21) Calculate $local\ top\text{-}k\ data$ from its own data and neighbors' $K$-skyband within search range
(22) $local\ top\text{-}k\ data \leftarrow local\ top\text{-}k\ data \cup lm\_data \cup me \cup mu\_data \cup ub$
(23) Sort $local\ top\text{-}k\ data$ by ascending order
(24) $lb' \leftarrow local\ top\text{-}k\ data[0]$
(25) $me' \leftarrow local\ top\text{-}k\ data[\lfloor k/2 \rfloor]$
(26) $j \leftarrow \lfloor k/2 \rfloor$
(27) while $me'.id < 0$ do
(28)     $j \leftarrow j + 1$
(29)     $me' \leftarrow local\ top\text{-}k\ data[j]$
(30) $ub' \leftarrow local\ top\text{-}k\ data[k-1]$
(31) $j \leftarrow k - 1$
(32) while $ub'.id < 0$ do
(33)     $j \leftarrow j + 1$
(34)     $ub' \leftarrow local\ top\text{-}k\ data[j]$
```

Algorithm 3: Filter construction.

is generated. As a result, this algorithm guarantees query transmission to all nodes holding data objects included in the top-$k$ result. After that, each query receiving node calculates the top-$k$ data objects from the received filter, supplemented {score, identifier} of upstream dataset, its own data objects, and its neighbors' *KSKY*, and then constructs its own filter (lines 21–34).

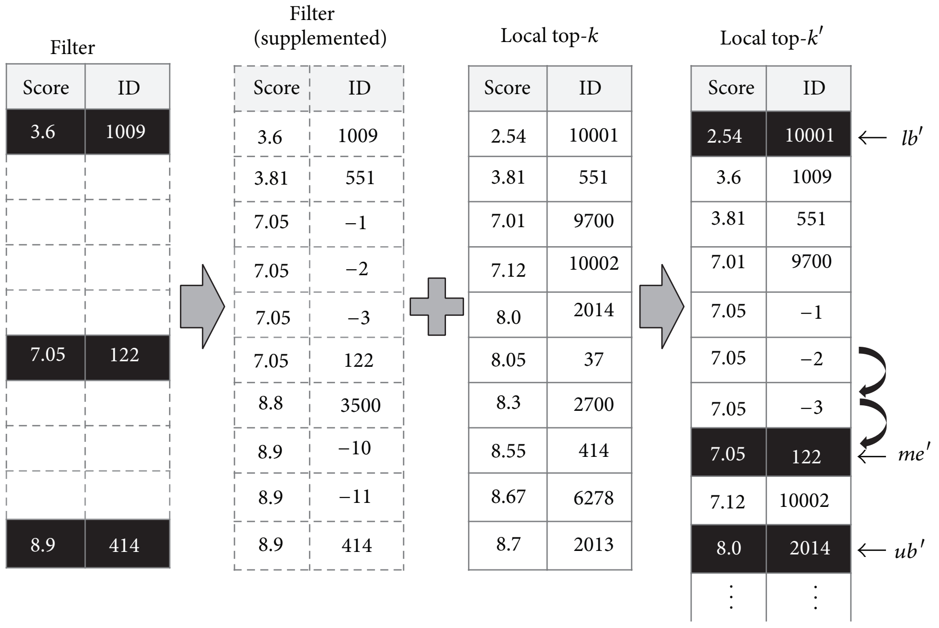
We illustrate an example of filter construction in Figure 3 where $k = 10$. A node, $N_p$, receives a query message attached with a filter (on the left) from node $N_q$. Suppose that there exists a node, $N_{p'}$, holding data objects whose scores and identifiers are $\{3.81, 551\}$ and $\{8.8, 3500\}$ in the common communication range of $N_p$ and $N_q$. $N_p$ can acquire this information from $KSKY_{p'}$. $N_p$ then supplements the missing information with *me* ($\{7.05, 122\}$) and *ub* ($\{8.9, 414\}$) and completes the filter. Finally, $N_p$ calculates the top-$k$ data objects (*local top-*$k'$ in Figure 3) from the filter, its own data objects, and its neighbors' *KSKY* (*local top-k* in Figure 3) and constructs its filter ($lb'$, $me'$, and $ub'$ in Figure 3).

*4.4. Top-k Query Processing.* In this subsection, we describe the message processing algorithm of ClusTo. ClusTo employs two kinds of query, *CtoC-query* (cluster to cluster query) and *data_acquisition query*. ClusTo transmits CtoC-queries over the search range and then transmits data_acquisition queries to nodes within the search range, holding data objects with scores not larger than the threshold.

A CtoC-query, $qc_i$, is represented as $qc_i = \langle \text{org}, \text{org\_position}, \text{sender\_id}, k, d, w, F, AT, CL \rangle$. $i$ is the identifier of the query. org is the identifier of the query originator node, $N_{org}$. org_position is the location information on $N_{org}$. $N_{org}$ specifies the number of requested data objects, $k$, a search range, $d$, and $w = \{w_1, w_2, \ldots, w_m\}$ and then constructs the filter, $F$. After that, $N_{org}$ stores all identifiers of cluster-heads ($\text{CH}_j$) and their corresponding next_nodes ($\text{next\_node}_j$) in its routing table in *AT* (=*AddressTuple*) such as $\{\{\text{CH}_0, \text{next\_node}_0\}, \{\text{CH}_1, \text{next\_node}_1\}, \ldots\}$. If $\text{next\_node}_j$ does not exist within the search range, $\{\text{CH}_j, \text{next\_node}_j\}$ is not stored in *AT*. *CL* (=*ClusterList*) is a list which contains

```
(1) if $N_p$ receives $q_q$ for the first time then
(2)     parent ← $q_q$.sedner_id
(3)     Set $F$ by Algorithm 3
(4)     threshold ← $q_q.F[2]$.score //ub.score
(5)     if $qc_q$ then
(6)         $CL \leftarrow q_q.CL$
(7)         for $\forall q_q.AT$ where $q_q.AT$.next_node $\neq N_p$ do
(8)             $CL \leftarrow CL \cup q_q.AT$.CH
(9)         $AT \leftarrow \emptyset$
(10)        for $\forall t \leftarrow$ RoutingTable.entry do
(11)            if $t$.CH is not included in $CL$ and
(12)            $t$.next_node is within $d$ from $N_{org}$ then
(13)                $AT \leftarrow AT \cup \{t$.CH, $t$.next_node$\}$
(14)        if $AT \neq \emptyset$ then
(15)            $qc_q \leftarrow \langle$org, org_position, $p, k, d, w, F, AT, CL\rangle$
(16)            Send $qc_q$ to nodes included in $AT$ as next_node
(17)        else
(18)            LeafFlag ← 1
(19)            Perform Algorithm 5
(20)    if $qd_q$ then
(21)        Reply data whose score is not more than threshold to parent
```

Algorithm 4: Handling query message, $q_q$.



Filter

| Score | ID |
|---|---|
| 3.6 | 1009 |
| | |
| | |
| | |
| | |
| 7.05 | 122 |
| | |
| | |
| | |
| 8.9 | 414 |

Filter (supplemented)

| Score | ID |
|---|---|
| 3.6 | 1009 |
| 3.81 | 551 |
| 7.05 | −1 |
| 7.05 | −2 |
| 7.05 | −3 |
| 7.05 | 122 |
| 8.8 | 3500 |
| 8.9 | −10 |
| 8.9 | −11 |
| 8.9 | 414 |

Local top-$k$

| Score | ID |
|---|---|
| 2.54 | 10001 |
| 3.81 | 551 |
| 7.01 | 9700 |
| 7.12 | 10002 |
| 8.0 | 2014 |
| 8.05 | 37 |
| 8.3 | 2700 |
| 8.55 | 414 |
| 8.67 | 6278 |
| 8.7 | 2013 |

Local top-$k'$

| Score | ID | |
|---|---|---|
| 2.54 | 10001 | ← $lb'$ |
| 3.6 | 1009 | |
| 3.81 | 551 | |
| 7.01 | 9700 | |
| 7.05 | −1 | |
| 7.05 | −2 | |
| 7.05 | −3 | |
| 7.05 | 122 | ← $me'$ |
| 7.12 | 10002 | |
| 8.0 | 2014 | ← $ub'$ |
| ⋮ | ⋮ | |

Figure 3: An example of filter construction where $k = 10$.

the identifiers of cluster-heads to which the query has been transmitted, so initially $CL = \emptyset$ in $N_{org}$'s query message. (If $N_{org}$ is a cluster-head, $CL = \{$org$\}$.) $N_{org}$ sends the query message to the next_nodes included in the $AT$. Algorithms 4 and 5 explain the procedure for a query receiving node, $N_p$. Algorithm 6 describes the procedure for reply messages. The query receiving node, $N_p$, transmits the query to cluster-heads (or next_nodes corresponding to them) except for the cluster-heads to which the query has been already transmitted (lines 6–13 in Algorithm 4). If next_node is out of the search range, $N_p$ does not transmit the query to the cluster (lines 9–13 in Algorithm 4). If $AT = \emptyset$, $N_p$ recognizes that it is a leaf node and sends a data_acquisition query to its neighboring nodes, within the search range, which hold data objects whose scores are not larger than the threshold (lines 18-19 in Algorithm 4).

A data_acquisition query, $qd_i$, is represented as $qd_i = \langle$org, org_position, sender_id, $k, d, w, F, A\rangle$. $A$ (=*Address-set*)

```
(1)  $A \leftarrow \emptyset$
(2)  if LeafFlag = 1 then
(3)      Put in all neighbors' identifiers, $N_r$ which fulfill all the following conditions to $A$
(4)      (i) $N_r$ is within $d$ from $N_{org}$
(5)      (ii) $N_r$ has data whose score is not more than threshold
(6)      (iii) $N_r$'s cluster head is not within $d$ from $N_{org}$
(7)  if $N_p$ is a cluster head then
(8)      Put in all cluster members' identifiers, $N_r$ which fulfill all the following conditions to $A$
(9)      (i) $N_r$ is within $d$ from $N_{org}$
(10)     (ii) $N_r$ has data whose score is not more than threshold
(11)     (iii) $N_r$ is not parent
(12) if $A \neq \emptyset$ then
(13)     $qd_q \leftarrow \langle$org, org_position, $p, k, d, w, F, A\rangle$
(14)     Send $qd_q$ to nodes included in $A$
(15) else
(16)     Reply data whose score is not larger than threshold to parent
```

Algorithm 5: Procedure by leaf nodes and cluster-heads.

```
(1)  /* $N_p$ receives $r_i$ */
(2)  if threshold > $r_i$.threshold then
(3)      threshold ← $r_i$.threshold
(4)  $D \leftarrow D \cup r_i.D$
(5)  if $N_p$ receives reply from all next_nodes inlcluded in $AT$ for the fist time then
(6)      if $N_p$ is a cluster head then
(7)          Perform Algorithm 5
(8)      else
(9)          $D' \leftarrow \{\forall$data $\in D \mid$ data.score $\leq$ threshold$\}$
(10)         $r_i \leftarrow \langle p, D'$, threshold$\rangle$
(11)         Send $r_i$ to parent
(12) else
(13)     $N_p$ receives reply from all next_nodes inlcluded in $A$
(14) $D' \leftarrow \{\forall$data $\in D \mid$ data.score $\leq$ threshold$\}$
(15) $r_i \leftarrow \langle p, D'$, threshold$\rangle$
(16) Send $r_i$ to parent
```

Algorithm 6: Handling reply message, $r_i$.

is a list of the identifiers of nodes to which $qd_i$ is transmitted. Leaf nodes and cluster-heads which have received reply messages from all next_nodes in their $AT$ send a data_acquisition query based on the conditions in lines 3–11 in Algorithm 5. Nodes receiving this message calculate the threshold from their own filter. ClusTo transmits CtoC queries as far as leaf nodes and then sends data_acquisition queries. This flow can make the threshold become a small value as possible and suppress unnecessary transmissions to noncontributors. Here, there is a case where although a cluster-head is outside of the search range, its cluster-members are within the search but near the border of the search range. The cluster-members cannot be targets of data_acquisition queries from their cluster-head. The problem is that even if they hold data objects included in the top-$k$ result, the query is not transmitted to them from their cluster-head. Therefore, ClusTo makes leaf nodes send the data_acquisition query to such cluster-members, according to the condition of line 6 in Algorithm 5, which guarantees the query transmission to them.

A reply message, $r_i$, is represented as $r_i = \langle$sender_id, $D$, threshold$\rangle$. sender_id is the identifier of the node which sends this message. $D$ includes the data objects with scores not larger than the threshold. Leaf nodes that do not need to send the data_acquisition query (lines 14–16 in Algorithm 5) and nodes receiving the data_acquisition query send a reply message to their parent. Reply receiving nodes update the threshold, and when received reply messages from all next_nodes in their $AT$ (and $A$ in the case of cluster-heads), they reply data objects with scores not larger than the threshold to their parent.

Query and reply receiving nodes send an ACK message to the message sender node for receipt acknowledgement. If the sender nodes do not receive an ACK message within a certain time, they retransmit the query (reply) message.

Following the above procedures, ClusTo can retrieve data objects included in the top-$k$ result while suppressing the number of nodes to which the query is transmitted. Recall that ClusTo designates as cluster-heads nodes holding data objects with high probability to be included in many top-$k$ results (Section 4.2). ClusTo has a great advantage, derived from its clustering, that the threshold gets a closer value to the exact value by cluster-heads. This is because each node transmits queries to the cluster-heads and they typically hold data objects whose scores can be smaller for any scoring functions. As a result, ClusTo can significantly suppress the number of nodes to which data_acquisition queries are transmitted, resulting in small traffic.

## 5. Simulation Experiments

In this section, we show the results of simulation experiments regarding the performance evaluation of our proposed method. For the simulation experiments, we used a network simulator, QualNet6.1 (Scalable Network Technologies: Creaters of QualNet Network Simulator Software (http://www.scalable-networks.com)).

### 5.1. The Dataset

*5.1.1. Intel Berkeley Lab Data.* We used dataset of wireless sensor networks collected by Intel Berkeley Research Lab (http://db.csail.mit.edu/labdata/labdata.html). The dataset consists of sensor readings collected from 54 nodes. We observed some missing and error sensor readings, so we selected a complete dataset of temperature, humidity, and light over a one-week (February 29 to March 6, 2004). In addition, we normalized the value of sensor readings between $[0, 250]$ and assume that the size of data objects is 128 [bytes] (size of metadata is 12 [bytes]).

*5.1.2. Synthetic Data.* We generated a synthetic $m$ dimensional dataset of uniform (UN), clustered (CL), and anticorrelated (AC) to evaluate ClusTo carefully. In the uniform dataset, the values of each metadata follow uniform distribution. In the clustered dataset, we draw a coordinate of its centroid of a cluster on the $m$-dimensional space $[0, value_{max}]^m$. The values of metadata on each dimension are independently follows Gaussian distribution with a variance of 30. The anticorrelated dataset is generated with reference to [16]. The characteristic of this dataset is that a data object tends to have high value on one dimension and low on the others. Each data object is $S$ [bytes] (metadata is $4\times m$ [bytes]), and we set $value_{max}$ as 250.

*5.2. Comparison Methods.* We compare the performance of ClusTo with ClusTo (OPT), 2-phase [15], and naïve method. In ClusTo (OPT), the query originator attaches an exact threshold (the $k$th lowest score in the given query condition) to the query message instead of a filter. Therefore, data objects not included in the top-$k$ results are never replied. Although this is impossible actually, we take this comparison method to investigate the performance of ClusTo compared with its optimal case. 2-phase is a method introduced in Section 3. The reason why we take this method as a comparison is that the assumption of this method is the most similar to our assumption, for example, wireless multihop network, and all nodes can generate queries. In the naïve method, the query originator broadcasts a query message attached with the $k$-lowest scores and their corresponding data identifiers. Each query receiving node calculates the top-$k$ data objects from the received scores and its own data objects and then broadcasts the query message attached with the those scores and identifiers. Leaf nodes set the threshold, as in ClusTo, and replies data objects with scores not larger than the threshold. This method is a special case of the method proposed in [14] and can strictly prune unnecessary data objects compared with the original method in [14] which transmits only a part of the $k$-lowest scores (i.e., the overhead can be much).

Table 1: Configuration of parameters.

| Parameter | Default | Range (type) |
|---|---|---|
| Number of nodes, $n$ | 300 | 200–500 |
| Value of $k$ | — | 1–15 |
| Data size, $S$ [bytes] | 128 | 32–256 |
| Dimensionality, $m$ | 3 | 2–5 |
| Data distribution | UN | CL, AC |
| Data update rate [%] | 30 | 0–100 |

*5.3. Simulation Model.* $n$ wireless nodes exist randomly in an area of 800 [m] × 800 [m]. Each node transmits messages and data objects using an IEEE 802.11b device whose data transmission rate is 11 [Mbps]. The transmission power of each node is determined so that the radio communication range becomes about 100 [m]. Packet losses and delays occur due to a radio interference. We distribute the dataset to every node and the number of data objects per node is 50 (i.e., the total number of data objects is $n \times 50$). Data objects are updated as time passes ($x$% of data objects are updated in the simulation).

We set $K$ as 15 and limit the number of reply retransmissions per node to 10. We investigated the accuracy of the query result within the limited number of reply retransmissions. Furthermore, we set $\alpha$ and $\beta$ in (4) as 0.35 and 100, respectively, based on some preliminary experiments. Each node generates a query message between a random interval within [60, 600] [sec] with random weighting. The number of requested data objects, $k$, follows Gaussian distribution (except for the impact of $k$ in Section 5.4) with a mean and variance of 5, and the range of $k$ is $[1, K]$. The search range, $d$, is calculated as $d = 50 \cdot j$ [m]. $j$ follows Gaussian distribution with a mean of 5 and a variance of 20, and the range of $d$ is $[2, 8]$. Table 1 shows the range and default setting of parameters in our experiments. We set the simulation time as 900 [sec] and evaluated the following criteria.

(i) *Initial overhead*: traffic defined as the total volume of messages sent in the clustering procedure.

(ii) *Data update overhead*: traffic defined as the total volume of messages sent during the simulation to inform updated $K$-skyband.

(iii) *Query overhead*: traffic defined as (the total volume of messages sent on top-$k$ query processing during the simulation)/the number of generated query.

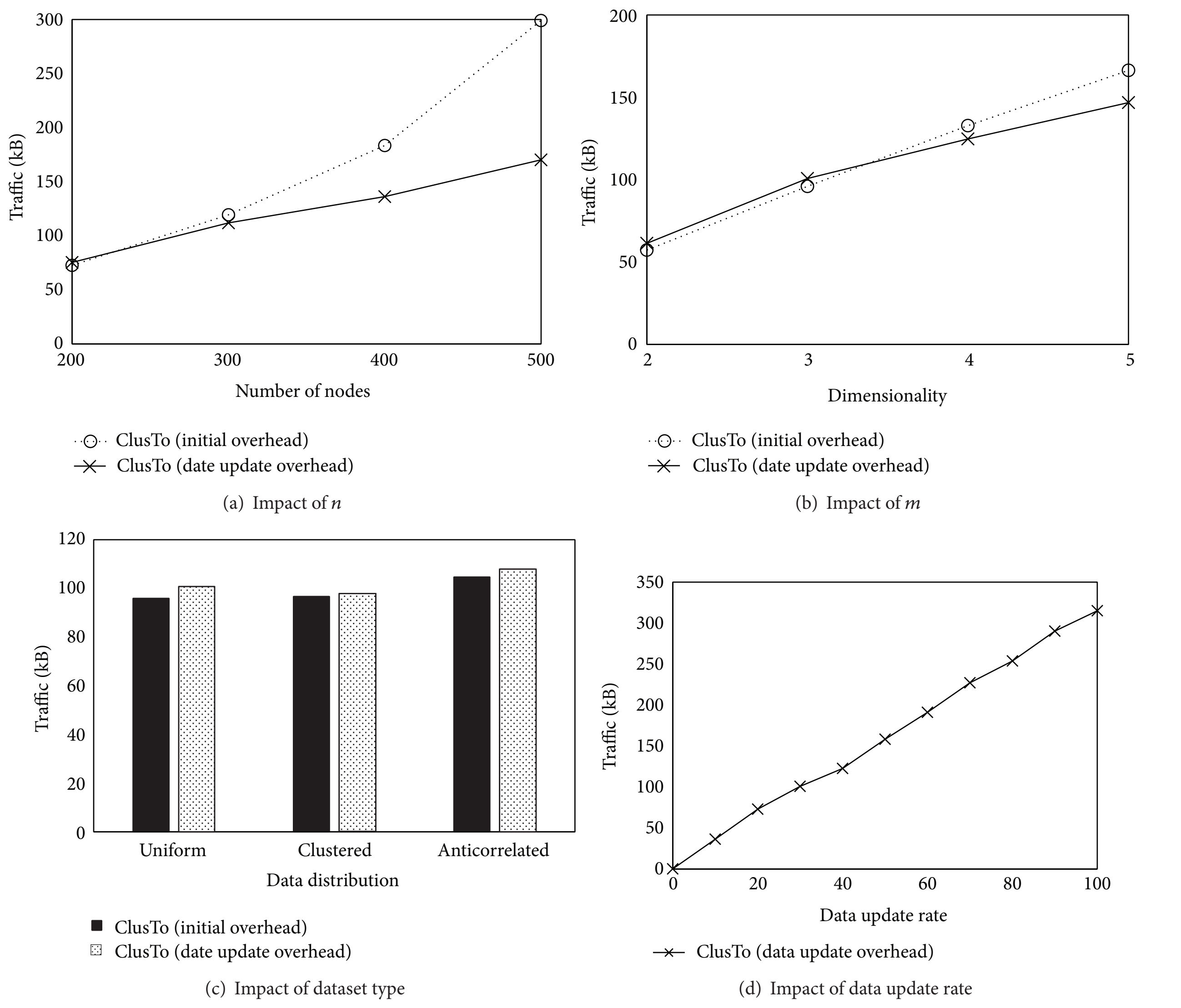


FIGURE 4: Initial overhead and data update overhead.

(iv) *Number of contacted nodes*: the average number of nodes to which a query message is transmitted.

(v) *Accuracy of query result*: MAP (*Mean Average Precision*) which is often used to calculate the accuracy with a rank. MAP averages accuracy of each query result, AP (*Average Precision*). AP and MAP are determined by the following equations:

$$AP_i = \frac{1}{k}\sum_{j=1}^{k}\frac{y}{j}\cdot e,$$
$$MAP = \frac{1}{querynum}\sum_{i=1}^{querynum} AP_i, \tag{7}$$

where $AP_i$ is the accuracy of the $i$th query result, $y$ is the number of collected data objects with the $j$-lowest scores included within the $j$-lowest correct answer set, $querynum$ is the total number of generated queries, and $e$ is determined by the following equation:

$$e = \begin{cases} 1 & (j\text{th answer is included in collected data}) \\ 0 & (\text{otherwise}). \end{cases} \tag{8}$$

Thus, $MAP$ rises as the query originator obtains data objects with smaller score.

(vi) *Delay*: the average of the elapsed time between the query originator generating a top-$k$ query and its acquisition of the result.

## 5.4. Simulation Results

*5.4.1. Initial Overhead and Data Update Overhead.* Firstly, we examined the initial overhead and data update overhead of ClusTo. Figure 4 shows the simulation results. In Figure 4(a),

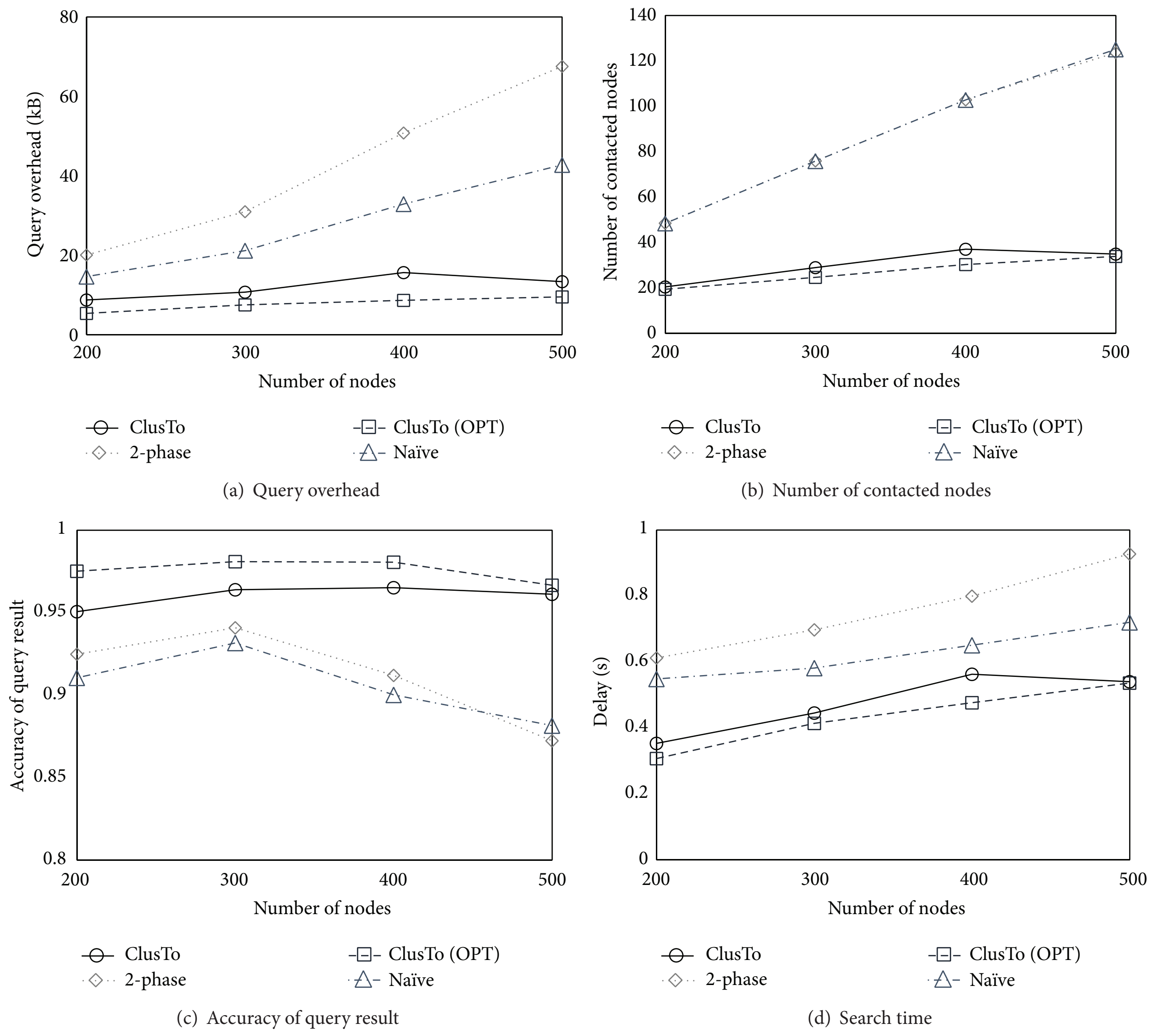


Figure 5: Impact of the number of nodes, $n$ (real dataset).

we used real dataset and examined the impact of $n$. In Figures 4(b), 4(c), and 4(d), we used the synthetic dataset and examined the impact of $m$, the dataset type, and the data update rate, respectively.

From Figure 4(a), we can see that both the initial overhead and the data update overhead increase as $n$ increases. This is to be expected because the number of $cf$ and $mn$ sent by nodes increase as $n$ increases. Also, the number of data objects in the network increases as $n$ increases, and this affects the overhead of updated $K$-skyband notification.

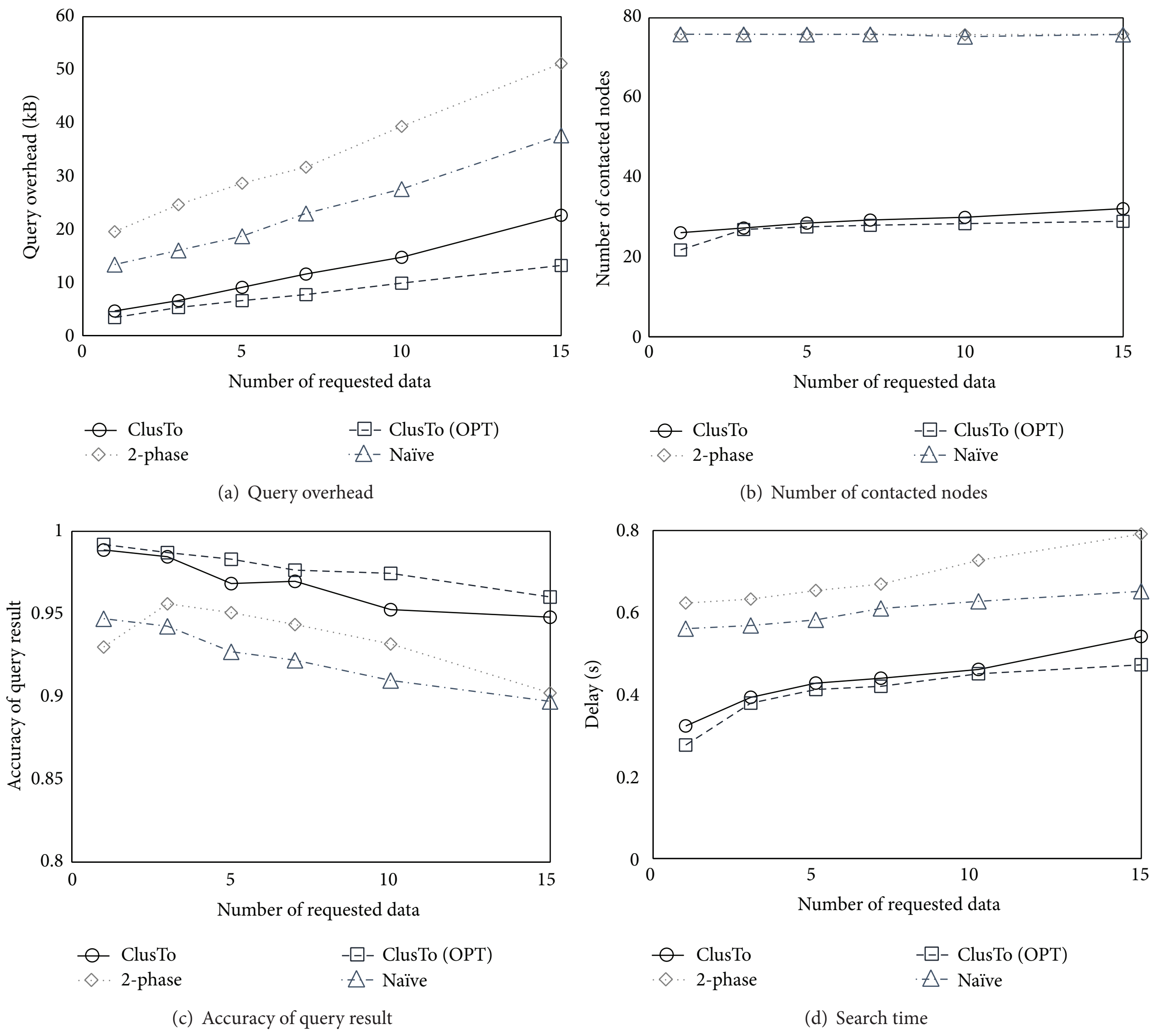
Figure 4(b) shows that both the initial overhead and the data update overhead increase as $m$ increases. A data object rarely dominates other data objects in the case of large $m$, so the number of data objects included in $K$-skyband increases. As a result, the size of $cf$ and a message which informs updated $KSKY$ increases.

From Figure 4(c), we can see that both overheads are large in the case of the anticorrelated dataset, where the number of dominated data objects is typically smaller than other datasets, resulting in the same impact as in the case of large $m$.

From Figure 4(d), we can see that the data update overhead increases as the data update rate increases. This is also instinctive because as many data updates occur, the $KSKY$ of each node is also updated. Hence, the number of message to inform the updated $KSKY$ increases. Note that the data update rate is irrelevant to the initial overhead.

*5.4.2. Top-k Query Processing.* Next, we examined the performance of each top-$k$ query processing method.

*Impact of n.* We examined the impact of the number of nodes, $n$. Figure 5 shows the simulation result with varying $n$. Figures 5(a), 5(b), 5(c), and 5(d) plot the measured performance in terms of query overhead, the number of contacted nodes, accuracy, and delay respectively. In this experiment, we used the real dataset.



(a) Query overhead

(b) Number of contacted nodes

(c) Accuracy of query result

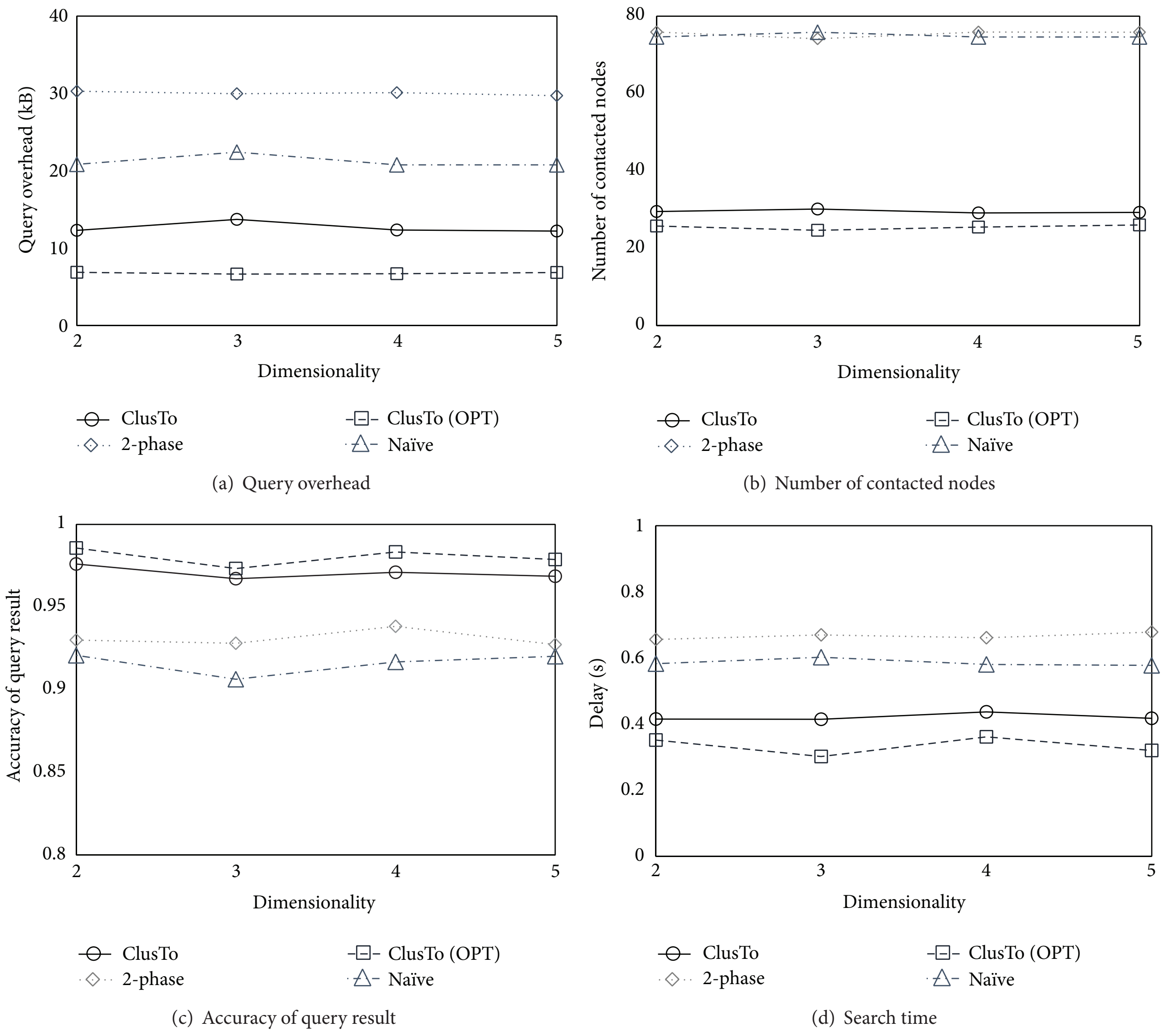
(d) Search time

Figure 6: Impact of the number of requested data objects, $k$ (real dataset).

From Figure 5(a), we can see that ClusTo performs top-$k$ query processing with small query overhead. In 2-phase and naïve method, query flooding generates much query overhead as $n$ increases. On the other hand, ClusTo routes queries to each cluster-head within the search range and then transmits data_acquisition queries to nodes holding data objects with scores not larger than the threshold, thereby suppressing the number of nodes involving in query processing. Thus, in ClusTo, the number of nodes that queries are transmitted does not change basically (or changes slightly) if $n$ increases. We can see this result in Figure 5(b). ClusTo significantly reduces the number of contacted nodes in comparison with 2-phase and naïve method. Note that, in the case where $n = 300$, the difference in query overhead between ClusTo and 2-phase is $31.0 - 10.7 = 20.3$ [KB]. In ClusTo, summation of the initial overhead and the data update overhead is 231.2 [KB] (= 119.3 + 111.9). Therefore, if top-$k$ query processing is performed ($231.2/20.3 = 11.4\cdots \fallingdotseq 12$) times in this setting, the total traffic of ClusTo can be less than that of 2-phase. This value, 12, is less than only 5% of the total number of nodes, so ClusTo is highly effective in cases where many nodes generate queries.

From Figure 5(c), we can see that ClusTo (as well as ClusTo (OPT)) outperforms the other methods. The query routing in ClusTo suppresses unnecessary query overhead and this prevents the decrease in the accuracy of query result caused by packet losses. ClusTo (OPT) keeps higher accuracy of query result than ClusTo. This is because ClusTo (OPT) replies only data objects included in the top-$k$ result, resulting in a reduction of packet losses more than ClusTo. Basically, the accuracy of query result of 2-phase and naïve method decreases as $n$ increases. Due to the query flooding in the methods, packet losses caused by message collisions frequently occur in the case where the number of nodes is large.

Figure 5(d) shows that the delay of ClusTo is shorter than that of 2-phase and naïve method. This is because ClusTo can retrieve the top-$k$ result without transmitting queries to



(a) Query overhead
(b) Number of contacted nodes
(c) Accuracy of query result
(d) Search time

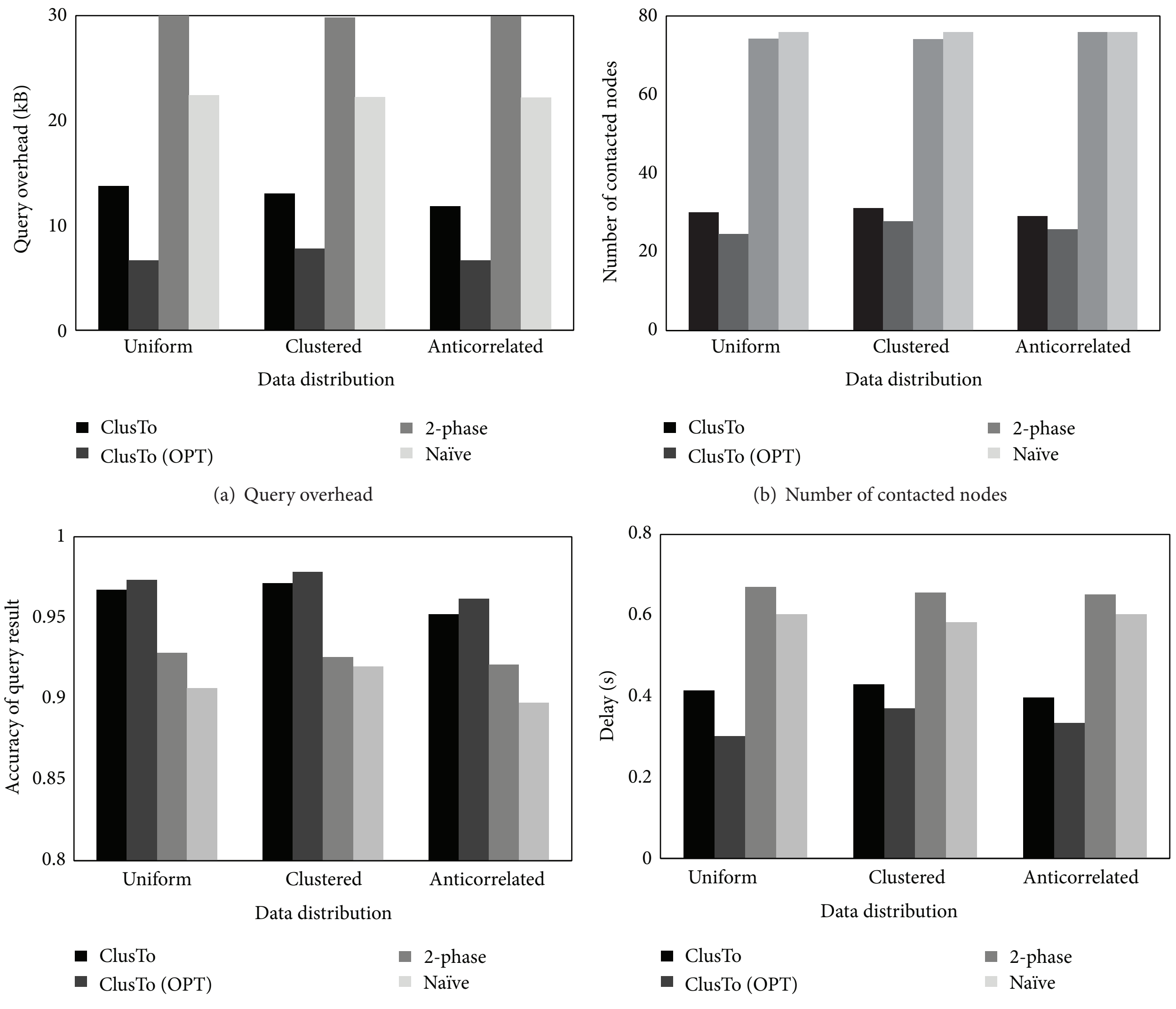
Figure 7: Impact of dimensionality, $m$ (synthetic dataset).

all nodes within the search range. Moreover, ClusTo sends a data_acquisition query to only nodes holding data objects with scores not larger than the threshold. This approach can reduce query overhead and packet losses, which suppresses the number of reply retransmissions, resulting in small delay. ClusTo (OPT) can reduce the delay further compared with ClusTo because ClusTo (OPT) sends a data acquisition query to fewer nodes than ClusTo due to the optimal threshold. The delay of 2-phase is longer than that of naïve method because of the two-time query flooding in each top-$k$ query processing.

*Impact of* $k$. We examined the impact of the number of requested data objects, $k$. Figure 6 shows the simulation result with varying $k$. Figures 6(a), 6(b), 6(c), and 6(d) plot the measured performance in terms of query overhead, the number of contacted nodes, accuracy, and delay, respectively. Also in this experiment, we used the real dataset.

From Figure 6(a), we can see that the query overhead in all methods increases as $k$ increases. This is because the number of replied data objects increases. When $k$ is large, the threshold in ClusTo becomes large and the number of nodes to which the data_acquisition query is transmitted increases (see Figure 6(b)). In addition, the increase in query overhead causes many packet losses; thus the accuracy in all methods decreases as $k$ increases as shown in Figure 6(c). ClusTo, however, keeps high accuracy of query result and small delay, as seen in Figure 6(d), compared with 2-phase and naïve method because ClusTo suppresses the number of transmitting messages as mentioned above. From the results of Figures 5(a) and 6(a), we can see that query flooding generates too much traffic and is inefficient for wireless multihop networks.

*Impact of m.* We examined the impact of the dimensionality, $m$. Figure 7 shows the simulation result with varying $m$.



(a) Query overhead

(b) Number of contacted nodes

(c) Accuracy of query result

(d) Search time

Figure 8: Impact of the dataset type (synthetic dataset).

Figures 7(a), 7(b), 7(c), and 7(d) plot the measured performance in terms of query overhead, the number of contacted nodes, accuracy, and delay, respectively. In the experiments from here, we used synthetic dataset because we can arbitrarily change the target parameter.

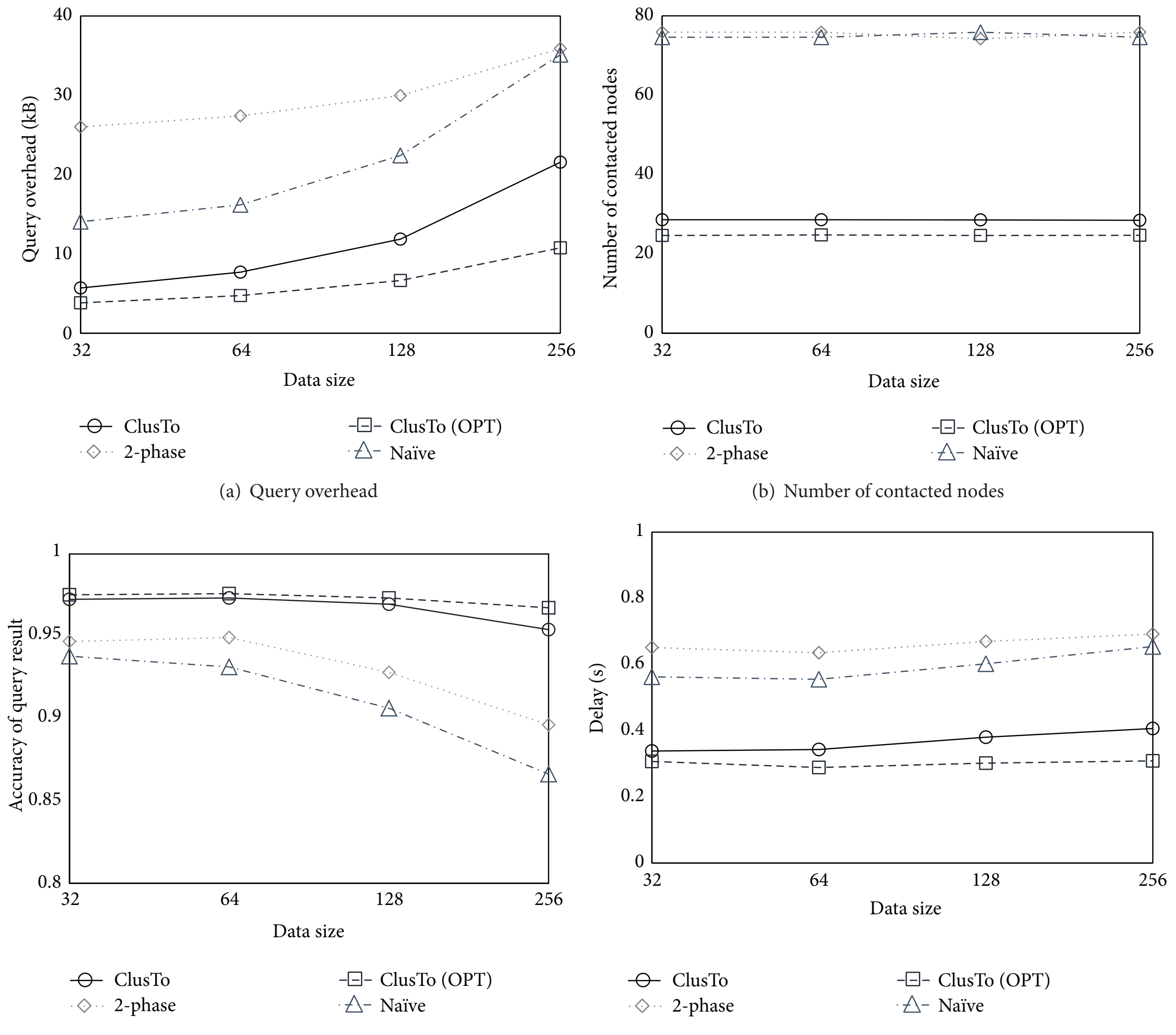
From Figure 7, we can see that the performance of ClusTo is basically independent of $m$. ClusTo transmits queries based on (clusters and) the threshold, that is, aggregating $m$-dimensional value to 1-dimension. Therefore, ClusTo has a great advantage in cases of high-dimensional datasets because the performance of ClusTo (like that of other methods) is not affected by $m$.

We can observe from Figure 7(a) that ClusTo reduces the query overhead by up to about 60% and 40% in comparison with 2-phase and naïve method, respectively.

As well as the result of Figure 7(a), the efficiency of ClusTo can be seen in Figures 7(b), 7(c), and 7(d). Figure 7(c) shows the high accuracy of query result of ClusTo. In any $m$, ClusTo outperforms 2-phase and naïve method because of its efficient query processing. Moreover, the accuracy of ClusTo is much the same with that of ClusTo (OPT), which implies the efficiency of the filter algorithm of ClusTo. We can observe from Figure 7(d) that ClusTo reduces the delay by up to about 40% and 30% in comparison with 2-phase and naïve method, respectively.

*Impact of Data Distribution*. We examined the impact of data distribution. Figure 8 shows the simulation result with respect to UN, CL, and AC. Figures 8(a), 8(b), 8(c), and 8(d) plot the measured performance in terms of query overhead, the number of contacted nodes, accuracy, and delay, respectively.

From these figures, we can see that ClusTo keeps high performance for any datasets. ClusTo forms clusters so that nodes holding data objects with high probability to be included in many top-$k$ results become clusterheads. Due to this effect, the number of nodes to which a data_acquisition query is transmitted can be suppressed as shown in Figure 8(b). As a result, ClusTo can reduce



(a) Query overhead

(b) Number of contacted nodes

(c) Accuracy of query result

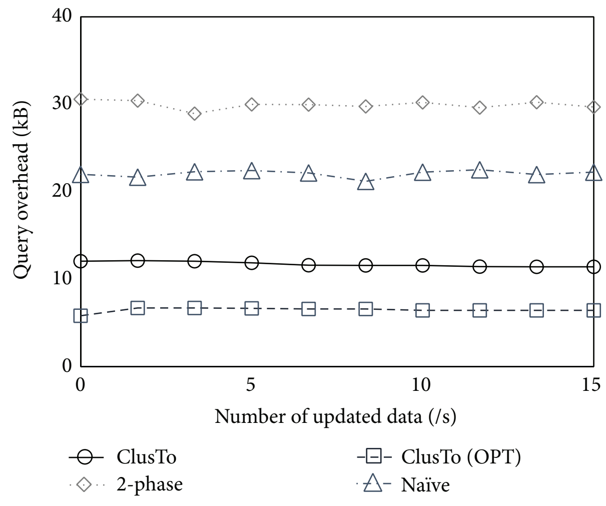
(d) Search time

FIGURE 9: Impact of the data size (synthetic dataset).

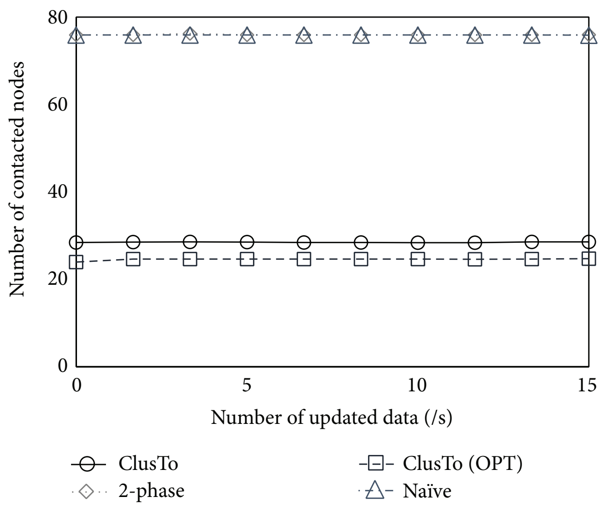
unnecessary query overhead and delay even if the dataset is anticorrelated as shown in Figures 8(a) and 8(d), which demonstrates the efficiency of our new clustering framework based on the skyline distance in (4).

*Impact of S.* We examined the impact of the size of data object, $S$. Figure 9 shows the simulation result with varying $S$. Figures 9(a), 9(b), 9(c), and 9(d) plot the measured performance in terms of query overhead, the number of contacted nodes, accuracy, and delay, respectively.

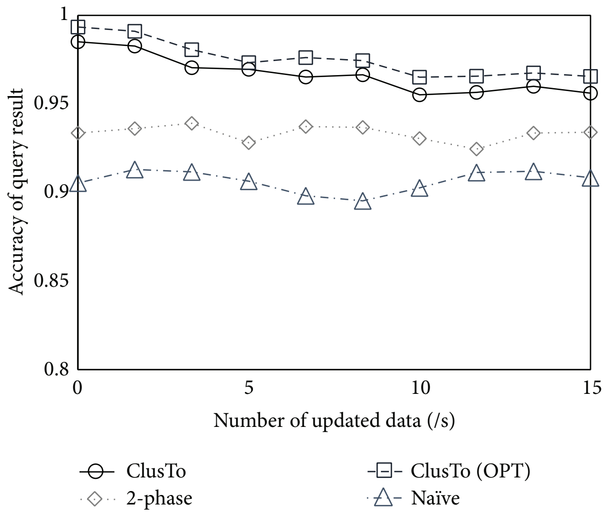
From Figure 9(a), we can see that the query overhead in all methods increases as $S$ increases. This is because the size of the reply message becomes large as $S$ increases. In the case of a large $S$, in particular, query overhead in all methods increases more because packet losses frequently occur and the number of reply retransmissions becomes large. ClusTo can suppress query overhead through its query routing and filter approach, resulting in a less number of nodes to access (Figure 9(b)), which prevents packet collisions as mentioned before. From Figure 9(c), we can see that ClusTo and ClusTo (OPT) keep high accuracy of query result while that of 2-phase and naïve method decreases as $S$ increases. Although 2-phase and naïve method suppress unnecessary data replies by 2-phase approach and filter approach, respectively, frequent message collisions caused by query flooding decrease the accuracy of query result. From Figure 9(d), we can see that ClusTo suppresses the delay even in the case of large $S$. Its query routing and filter approach, described above, suppress the number of nodes to which queries are transmitted. This reduces packet losses, resulting in less number of reply retransmissions. In contrast, the delay in naïve method is long and is similar to that of 2-phase even though this method processes a top-$k$ query in a single phase. In this method, more unnecessary data objects are replied than 2-phase, and this increases unnecessary query overhead and packet losses. As a consequence, the number of reply retransmissions increases.



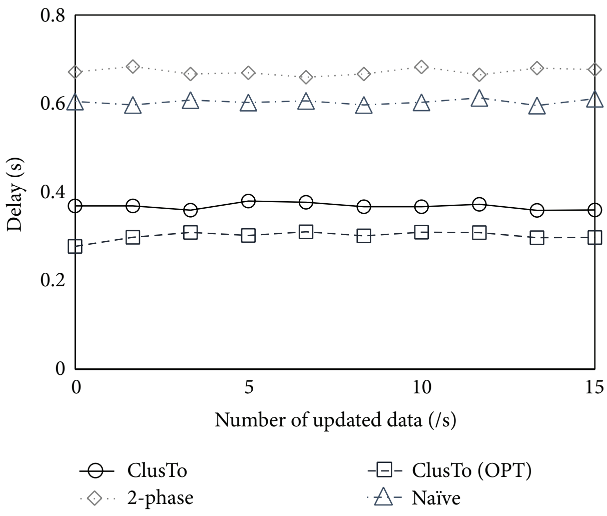
(a) Query overhead



(b) Number of contacted nodes



(c) Accuracy of query result



(d) Search time

FIGURE 10: Impact of the data update rate [/sec] (synthetic dataset).

*Impact of $x$.* In sensor networks and some other applications, data objects are possibly updated, so we examined the impact of data update rate to show the robustness of ClusTo for data update. Figure 10 shows the simulation result with varying data update rate. Figures 10(a), 10(b), 10(c), and 10(d) plot the measured performance in terms of query overhead, the number of contacted nodes, accuracy, and delay, respectively. In Figure 10, we rescaled the data update rate [%] in the simulation to the number of updated data [/sec].

From Figure 10, we can see that the data update rate does not affect the criteria in 2-phase and naïve method. This is because these methods do not maintain any information on data objects of neighboring nodes. In ClusTo, on the other hand, nodes maintain the information on only their neighboring nodes. If a data update occurs, the nodes holding the data objects broadcast the updated *KSKY* to their neighboring nodes, ensuring that the neighboring nodes keep correct information on their $K$-skyband. Nodes to which CtoC queries are transmitted do not change even if a data update occurs because the queries are transmitted based on routing tables. Furthermore, the number of nodes to which data_acquisition queries are transmitted does not increase or decrease even if a data update occurs. More specifically, even if a data update occurs, the target nodes for data_acquisition queries are simply changed to other nodes. As a result, the query overhead, the number of contacted nodes, and the delay are constant, and this is the advantage of our proposed method for the case where many data objects are updated.

From Figure 10(c), we can see that the accuracy of query result in ClusTo and ClusTo (OPT) slightly decreases as the number of updated data per sec increases. In ClusTo, if the number of updated data increases, the chance of neighboring nodes being formed of the updated *KSKY* increases. Consequently, frequent transmissions of such messages cause many message collisions in top-$k$ query processing. This leads to a decrease in the accuracy of query result. However, the accuracy in ClusTo is still high even in the case where all the data objects are updated.

## 6. Conclusion

In this study, we proposed ClusTo, which realizes efficient multidimensional top-$k$ query processing in wireless multihop networks, where users can specify a monotone scoring function and a search range. The proposed method performs effective clustering with skyline, which streamlines query routing and filter updating. Furthermore, ClusTo can construct more accurate filter by considering with $K$-skyband of neighboring nodes, which can set a threshold as closer value to the exact value. After routing queries over the search range, the queries are transmitted to nodes holding data objects whose scores are not larger than the threshold. This procedure suppresses unnecessary query transmissions to nodes which do not contribute to top-$k$ data retrieval. Our simulation results show that ClusTo achieves efficient top-$k$ query processing in terms of query overhead and delay as well as highly accurate query result even for limited reply retransmissions.

In this study, we assumed a wireless static network whose network topology does not change. However, in some applications, nodes may not simply move locally, which provokes link disconnections between nodes. In ClusTo, because each node constructs a filter and routes queries based on the information on neighboring nodes' position and data objects, such network topology change presents a challenging problem. We therefore plan to tackle this problem by extending ClusTo to adapt to such dynamic environments.

## Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

## Acknowledgments

This research is partially supported by the Grant-in-Aid for Scientific Research (B) (24300037) of MEXT and JSPS Fellows (26-4907) of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

## References

[1] Y.-C. Chung, I.-F. Su, and C. Lee, "Supporting multi-dimensional range query for sensor networks," in *Proceedings of the 27th International Conference on Distributed Computing Systems (ICDCS '07)*, pp. 35–43, Toronto, Canada, June 2007.

[2] Z. Gong, G.-Z. Sun, J. Yuan, and Y. Zhong, "Efficient top-$k$ query algorithms using $k$-skyband partition," in *Proceedings of the International Conference on Scalable Information Systems*, pp. 288–305, 2009.

[3] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "Prefer: a system for the efficient execution of multi-parametric ranked queries," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 11–20, May 2001.

[4] D. Xin, J. Han, H. Cheng, and X. Li, "Answering top-k queries with multi-dimensional selections: the ranking cube approach," in *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06)*, pp. 463–474, September 2006.

[5] R. Akbarinia, E. Pacitti, and P. Valduriez, "Reducing network traffic in unstructured P2P systems using Top-$k$ queries," *Distributed and Parallel Databases*, vol. 19, no. 2-3, pp. 67–86, 2006.

[6] W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden, "Progressive distributed top-k retrieval in peer-to-peer networks," in *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, pp. 174–185, April 2005.

[7] P. Cao and Z. Wang, "Efficient Top-K query calculation in distributed networks," in *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC '04)*, pp. 206–215, July 2004.

[8] S. Michel, P. Triantafillou, and G. Weikum, "KLEE: a framework for distributed top-k query algorithms," in *Proceedings of the31st International Conference on Very Large Data Bases (VLDB '05)*, pp. 637–648, September 2005.

[9] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis, "On efficient top-k query processing in highly distributed environments," in *Proceedings of the International Conference on Management of Data (SIGMOD '08)*, pp. 753–764, June 2008.

[10] H. Jiang, J. Cheng, D. Wang, C. Wang, and G. Tan, "Continuous multi-dimensional top-k query processing in sensor networks," in *Proceedings of the IEEE INFOCOM*, pp. 793–801, April 2011.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: a tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[12] M. Wu, J. Xu, X. Tang, and W.-C. Lee, "Top-k monitoring in wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 7, pp. 962–976, 2007.

[13] D. Amagata, Y. Sasaki, T. Hara, and S. Nishio, "A robust routing method for top-k queries in mobile ad hoc networks," in *Proceedings of the 14th International Conference on Mobile Data Management (MDM '13)*, pp. 251–256, June 2013.

[14] R. Hagihara, M. Shinohara, T. Hara, and S. Nishio, "A message processing method for top-k query for traffic reduction in Ad Hoc networks," in *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware (MDM '09)*, pp. 11–20, May 2009.

[15] Y. Sasaki, T. Hara, and S. Nishio, "Two-phase top-k query processing in mobile ad hoc networks," in *Proceedings of the 14th International Conference on Network-Based Information Systems (NBiS '11)*, pp. 42–49, Tirana, Albania, September 2011.

[16] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of the 17th International Conference on Data Engineering (ICDE '01)*, pp. 421–430, April 2001.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 41–82, 2005.

[18] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-based representative skyline," in *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE '09)*, pp. 892–903, April 2009.

[19] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Computing Surveys*, vol. 40, no. 4, Article ID 1391730, 2008.

[20] B. Chen, W. Liang, R. Zhou, and J. X. Yu, "Energy-efficient top-k query processing in wireless sensor networks," in *Proceedings*

*of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*, pp. 329–338, Toronto, Canada, October 2010.

[21] J. Niedermayer, M. A. Nascimento, M. Renz, P. Kröger, and H.-P. Kriegel, "Exploiting local node cache in top-k queries within wireless sensor networks," in *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '10)*, pp. 434–437, November 2010.

[22] A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang, "A sampling-based approach to optimizing top-k queries in sensor networks," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*, pp. 68–78, April 2006.

[23] X. Liu, J. Xu, and W.-C. Lee, "A cross pruning framework for Top-k data collection in wireless sensor networks," in *Proceedings of the 11th IEEE International Conference on Mobile Data Management (MDM '10)*, pp. 157–166, May 2010.

[24] B. Chen, W. Liang, and G. Min, "Top-*k* query evaluation in sensor networks with the guaranteed accuracy of query results," in *Proceedings of the 22nd International Conference on Database and Expert Systems Applications (DEXA '11)*, pp. 156–171, Toulouse, France, 2011.

[25] S.-J. Tang, X. Mao, and X.-Y. Li, "Efficient and fast distributed top-k query protocol in wireless sensor networks," in *Proceedings of the 19th IEEE International Conference on Network Protocols (ICNP '11)*, pp. 99–108, October 2011.

[26] L. Zou and L. Chen, "Dominant graph: an efficient indexing structure to answer top-K queries," in *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE '08)*, pp. 536–545, April 2008.

[27] Y. Sasaki, R. Hagihara, T. Hara, M. Shinohara, and S. Nishio, "A top-k query method by estimating score distribution in mobile ad hoc networks," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA '10)*, pp. 944–949, April 2010.