PAPER

Threshold-Based Distributed Continuous Top-k Query Processing for Minimizing Communication Overhead

Kamalas UDOMLAMLERT[†a)], ***Nonmember***, **Takahiro HARA**[†], ***Member***, ***and*** **Shojiro NISHIO**[†], ***Fellow***

SUMMARY In this paper, we propose a communication-efficient top-k continuous query processing method on distributed local nodes where data are horizontally partitioned. A designated coordinator server takes the role of issuing queries from users to local nodes and delivering the results to users. The final results are requested via a top-k subscription which lets local nodes know which data and updates need to be returned to users. Our proposed method makes use of the active previously posed queries to identify a small set of needed top-k subscriptions. In addition, with the pre-indexed nodes' skylines, the number of local nodes to be subscribed can be significantly reduced. As a result, only a small number of subscriptions are informed to a small number of local nodes resulting in lower communication overhead. Furthermore, according to dynamic data updates, we also propose a method that prevents nodes from reporting needless updates and also maintenance procedures to preserve the consistency. The results of experiments that measure the volume of transferred data show that our proposed method significantly outperforms the previously proposed methods.

key words: *top-k queries, skyline queries, continuous queries, multidimensional databases, distributed systems*

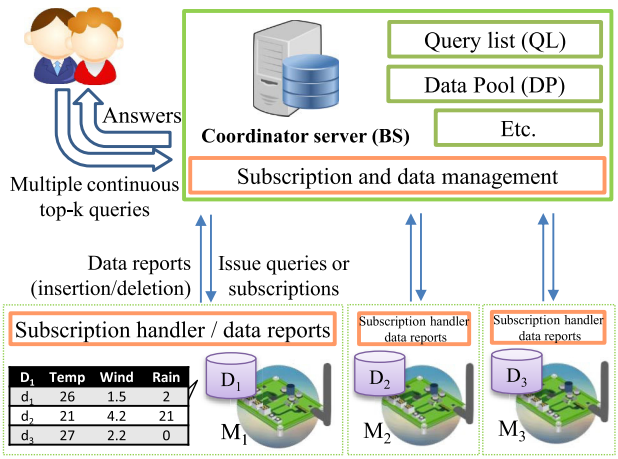


Fig. 1 An example of the assumed monitoring system

1. Introduction

Nowadays, massive data are continuously generated mostly in distributed fashion such as sensor data. These data can be associated with multiple dimensions of nominal, ordinal or numeric (interval) attributes. Apart from the large volume of data, these data are quite fast, i.e., some generated data can be replaced with newer generated data or can be expired.

A monitoring system that executes continuous queries plays an important role to detect and capture important incidents analyzed by those data especially in applications of environmental monitoring and disaster monitoring. Despite the massive amount of data, the monitoring system needs to display only some attentive data in real time because users or experts would be interested in only a few best data objects (say k objects) at a time, i.e., continuous top-k query processing. Basically, the ranks of data objects can be determined by their scores calculated by a scoring function (user preference) on numeric quantitative attributes

In this paper, we assume a general monitoring system depicted in Fig.1. Multiple users can register their top-k queries with different preferences into a single coordinator server (BS). The distributed nodes (M_i) consequentially generates a data object consisting of multiple numerical attributes or invalidates the previously generated data object. The coordinator server, acting as the interface between users and many distributed nodes, processes multiple continuous queries on behalf of many users and keeps delivering the latest top-k answers to those users according to the data changes (insertion or deletion). The components in BS and M_i will be explained in detail in the latter section.

However, the problem in this scenario is that, to correctly deliver or notify the final top-k answers to end-users, the coordinator server must aggregate sufficient data objects from the distributed nodes. A naive method is to let the coordinator server periodically issue many traditional distributed top-k snapshot queries at every time tick (proactive scheme). However, it is difficult to define an appropriate query frequency, i.e., the duration between time ticks. Setting too high frequency can lead to many redundant data objects returned while setting too low frequency can make some users miss out important up-to-date data. Another naive method is to aggregate all local data objects at the initialization phase as well as every data update to the coordinator server (reactive scheme). This guarantees that, for any queries, the aggregated data objects giving the k-highest scores are top-k answers, but this approach possibly transfers a lot of unused data objects, i.e., non-top-k objects.

From the viewpoint of green IT, it is also highly important to reduce the energy consumption according to communication cost which refers to the total amount of transferred query messages and data objects especially in sensi-

Manuscript received September 16, 2015.
Manuscript publicized November 11, 2015.
[†]The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.
a) E-mail: kamalas.u@ist.osaka-u.ac.jp
DOI: 10.1587/transinf.2015EDP7377

tive battery-constrained systems like sensor networks.

To solve the above problems, our proposed method adopts threshold-based top-k query processing in a reactive scheme and introduces techniques which help saving communication overhead as follows; (1) The coordinator server constructs a top-k subscription denoted by a scoring function and an estimated score as a threshold for each query. These top-k subscriptions are sent to local nodes to inform them of the scope of promising data objects, so they are able to send only some necessary data back to the coordinator server. (2) When handling a large number of continuous queries, it is inefficient to rigorously pass to every node a large list of all requests or subscriptions. We make use of the maintained queries as materialized views at the coordinator server to identify a small set of dominating queries to be informed to nodes. Moreover, some queries can possibly be answered by using those views (previously posed queries), so the coordinator server does not need to pay additional communication cost for issuing queries or retrieving top-k answers that incur wasteful transmission. (3) Our proposed method can avoid sending some subscriptions to local nodes whose promising data are still unlikely to be included in the top-k answers. This results in saving cost of issuing subscriptions.

In summary, the contributions of this paper are as follow:

- We formulate the requirements of multidimensional continuous top-k query processing in distributed environments. Then, we propose an efficient method that satisfies the requirements and takes the actual user's preference into account.
- We design a mechanism to minimize unnecessary data traffic and the maintenance cost of the proposed method in response to data and query updates.
- We conduct some experiments in various settings to show that our proposed method has lower communication overhead than competitive methods.

2. Related Work

Top-k query processing has been widely researched and gained a lot of attentions in research of databases. Initially there have been many studies on snapshot top-k processing in centralized databases [1]–[4]. Besides, in the research field of distributed systems, there have also been a number of proposed techniques which are favorable for various assumptions and requirements. For the assumption of data models, the first category is called vertically partitioned datasets which can be seen in [5]–[7], and another category is called horizontally partitioned datasets. The latter category is matched with our assumed environment. In [8], the authors proposed a method to acquire the top-k results in a P2P network where the dataset is vertically partitioned, and super-peers take roles of processing and indexing. However this method is not efficient when directly deploying on our constraints where we focus on more complicated multidimensional top-k query processing.

One technique for efficient top-k query processing is to answer top-k queries by using materialized views. In the case of centralized processing, to access a massive raw database is considered too costly. Instead of accessing the entire database relations every time, in [9], the authors proposed LPTA utilizing the answers of previous-posed top-k queries called views. LPTA based on linear programming determines whether the existing views are sufficient to respond to a new top-k query. In [10], the authors proposed algorithms of maintenance of a large number of top-k materialized views for 2-dimensional data. The existing views are indexed in hierarchy paths for efficient view updates, and views to be updated are judged by using those hierarchy paths. However, the hierarchy paths must be reconstructed on every update. In the case of distributed processing, ARTO [11] is a cached-based distributed top-k query processing method. Due to the existence of cache of the previous results, ARTO constructs a remainder query to fetch only the remaining data from horizontally partitioned databases. In our work, we also deploy this concept to identify whether the available data objects at the coordinator server is sufficient enough to answer some queries immediately.

In Wireless Sensor Networks (WSNs) where data are sent by multi-hop relays to the sink or the server, the goal is to minimize the communication cost for data transmission and prolong the network lifetime. In [12], the authors basically presented an in-network aggregation technique by sending only top-k answers from every node for each epoch. In [13], the algorithm called FILA constructs filters for nodes and binds those filters to nodes. Data which are filtered by a filter will not be sent to a sink. Most of these works assumed only single dimensional data and one sensor reading at a time. In [14], the authors proposed a method to aggregate top-k answers in WSNs by constructing a top-k filter utilizing a dominant graph which was discussed in [3] as a data structure. Even though our research focuses on general distributed systems, this work proposed in WSNs is comparable to our work because both focus on multidimensional and continuous top-k queries. However, this method does not take the actual scoring function into account, and it assumes each data object's expiration time is known (time-frame model) (See Sect. 5.1.2).

Continuous top-k query processing is also found in [15]–[18]. In [15], they firstly introduced the concept of distributed top-k monitoring supporting a sum of a single attribute monitoring query across vertically partitioned databases. [16] addresses multiple continuous top-k queries over data streams in centralized databases by aiming at reducing CPU cost. In [17], the authors proposed a real-time publish/subscribe model to perform continuous top-k query processing, while our problem is to reduce the communication cost of monitoring top-k answers on high-update horizontally partitioned databases. The works above are different from ours because they assume continuous queries on

data streams over sliding windows or time-frame models. Our work is much closer to [18] according to the problem definition, but this work focuses on centralized databases which the performance is measured by execution time.

Apart from a top-k query, a skyline query has been extensively researched both in centralized [19] and distributed fashions [20]. Because this type of query is related to the study of data dominance in multidimensional space, the utilization of skylines is also known as effective for top-k query processing. Specifically, skylines can be indexed and utilized as effective routing metrics. In [21], the authors assumed a super-peer architecture and make use of K-skyband cached at super-peers, and super-peers decide the best sequence of other super-peers to be requested. K-skyband is a set of data objects that are dominated by at most $K-1$ other objects. Basically, the data that belong to K-skyband can sufficiently answer all monotonic top-k queries where $k \leq K$. In [22], the proposed method called DiTo designates a server to aggregate skylines of other servers in a P2P network and derives benefits from that knowledge to iteratively send top-k query requests to the most promising nodes first. As a result, the number of disturbed nodes is minimized, but the number of iterations to finish the processing in the worst case is up to k, and this approach does not take continuous queries into account.

3. Preliminaries

3.1 Environment and Data Model

The distributed network consists of $N+1$ nodes including one base station (BS) and N local nodes ($M_1, M_2, \ldots, M_N$) called nodes for short. BS takes the role of query issuing and data aggregation. BS can be seen as a centralized coordinator server to relay and deliver the final result to end-users. Each node logically connects and collaborates with BS only. Hence, among local nodes, they share nothing. In this work, without loss of generality, we assume that the topology is unchanged and reliable. Figure 1 shows an example of a topology consisting of 3 nodes deployed in the network. A data object $d_j \in D_i (1 \leq i \leq N, 1 \leq j \leq |D_i|)$ held by a node i has a fixed size associated with its identifier (m-dimensional *numerical* attributes) which can be represented as a data tuple $d_j = (d_j[1], d_j[2], \ldots, d_j[m])$. The data objects are horizontally partitioned among nodes, i.e. a node holds only a partial set D_i of entire data objects in the system D ($D_i \subseteq D$). The data objects are arbitrarily inserted and removed at nodes as time passes.

3.2 Multidimensional Top-k Query

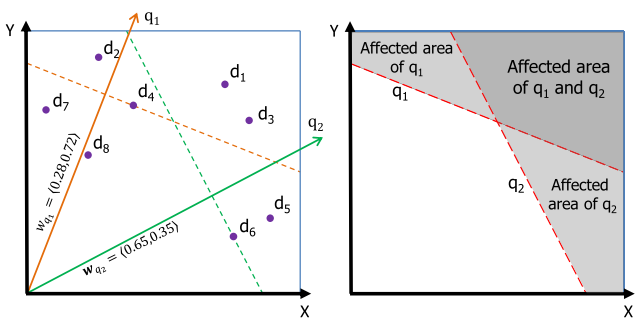
A multidimensional top-k query $qk = (sf, k)$ from a user is defined by a scoring function sf which is a monotonic function, and a parameter k. In this paper we focus on linear combination functions such that the score of a data item d_i is $sf(d_i) = \sum_{j=1}^{j=m} w_j \cdot d_i[j]$ where w_j stands for a positive weighting at j-th dimension, i.e., the degree of attention of a user over j-th dimension. A vector of weightings is denoted by $\mathbf{w} = \langle w_1, w_2, \ldots, w_m \rangle$. The summation of weighting of all dimensions should be equal to 1, but in fact, represented as a vector, only its direction affects the consequent result. A scoring function is monotonic that is if $d_a[i] \leq d_b[i]$ for all $1 \leq i \leq m$, then $sf(d_a) \leq sf(d_b)$. The value k defines the number of desired data objects.

It is noted that our query model does not include a user's location as one of the system's input, so the attribute must be *user-independent*, i.e., every user recognize the same values of attributes for each data object. In addition, we assume that the attribute must be quantitative attributes. For attributes of locations, either latitude or longitude may not be appropriate to be used solely as an attribute because they are not quantitative attributes. However, these attributes (latitude or longitude) can be used for calculating distances instead. Nevertheless, the distance from each user to the objects is not eligible while user-independent distances, e.g., distance to the nearest emergency dispatcher (which is static for evaluating the risks), distance to the nearest train station (which is static for evaluating properties) can be used. In this work, we assume that users prefer higher values, so in the case of distances, we may use 1/distance as an attribute instead.

In fact, top-k answers and non-top-k answers are easily divided by giving the query and the actual threshold. The actual threshold is k-th highest score of entire data objects distributed in the network (D). However, the actual threshold is difficult to know, because BS does not have entire data to rank all data objects. The threshold which is estimated is referred to *the threshold* whether it is actual. A query which is already attached with the threshold is represented as $q = (qk, ths) = (sf, k, ths)$.

Top-k answers of query q denoted by TOP_q is a set of data objects which contains only k data objects having best scores among data objects according to the scoring function $q.sf$ such that $TOP_q \subseteq D$ and $|TOP_q| = k$. Therefore, $\forall d_i \forall q (d_i \in TOP_q \rightarrow (q.sf(d_i) \geq q.ths))$.

In Fig.2a, the final answers of the top-4 query q_1 ranked by their scores are d_1, d_2, d_3 and d_4 respectively, and the threshold $q_1.ths = q_1.sf(d_4)$. Similarly, the answers of top-4 query q_2 ranked by their scores include d_3, d_1, d_5 and d_6



(a) The scoring vectors and their thresholds

(b) The affected area of the queries in Fig.2(a)

Fig. 2 Example of two top-4 queries including q_1 and q_2

respectively while the threshold $q_2.ths = q_2.sf(d_6)$. We define the area where the points have scores higher than the threshold ($q.sf \geq q.ths$) as the affected area, because any data that fall in this area definitely affect the query's threshold, and also define the line between those two areas as the threshold line ($q.sf = q.ths$). The example in Fig.2b shows the affected area and the threshold lines which are formed by 2 queries in Fig.2a. Only data updates that occur inside the affected area will affect the final results and the changes of the thresholds.

3.3 Continuous Top-k Query

A continuous query is different from a snapshot query that a user wishes to continuously monitor on the latest final answers of the query. The naive solution for this requirement is to periodically query for answers, but it is not efficient because a lot of redundant data objects will be retrieved each time. For this aim, the reactive method, informing a potential update as necessary, is more promising. Our proposed method assumes two important components stored at BS; *the query list* (QL) and *the data pool* (DP). QL records active queries from users, and DP maintains previously retrieved valid data objects at BS. We try to preserve that available data objects in DP must sufficiently satisfy all queries in QL.

3.4 Top-k Subscription

The top-k subscription of a query q (S_q for short) is denoted by 2 components of query q; a scoring function ($q.sf$) and a threshold ($q.ths$). $S_q = (sf, ths)$ is issued to local nodes in order to request the data objects whose scores surpass the threshold regarding to the scoring function back to BS. Nodes M_i also store received subscriptions in their own subscription list SL_i. In order to deliver the precise top-k answers of a continuous query q to end-users, the threshold indicated in the subscription must guarantee that the total number of data objects whose scores surpass the threshold is not less than $q.k$ ($|\{d_i \in D | S_q.sf(d_i) \geq S_q.ths\}| \geq q.k$). BS requests all those data objects to be stored in DP. As a result, the aggregated data objects in DP giving k best scores are the answers of the top-k query.

Due to the horizontally partitioned dataset, BS does not know the complete view of global data distribution. As a result, to acquire the actual threshold (k-th score of entire data objects D) is difficult. In this paper, we proposed the threshold estimation algorithm to achieve the near actual threshold.

Due to data updates, these thresholds possibly change and break the consistency. The maintenance of DP and QL as well as SL is required. We will describe this in Sect. 4.

3.5 Skylines and Their Utilization

Skyline operator was first proposed in [19] which became a type of queries and was extended to apply to many purposes. In our proposed method, we utilize the characteristics of the skyline operator for several benefits. Inspired by the skyline utilization in [22], indexing local nodes' skylines at BS enables BS to selectively request the data from the minimal number of nodes for distributed top-k query processing. Due to the threshold-based procedures, apart from the initialization of aggregating local skyline, we can avoid flooding that causes a huge amount of traffic in the large scale (explained in Sect. 4). It is noted that a skyline cardinality increases exponentially with dimensionality; however, the initialization is done once and it can be used in the long term, so we maintain the skyline in response to updates (less frequent).

Definition 1: Given a set of data points P, a skyline query of P returns data points $skyline(P)$, such that $p \in skyline(P)$ is not dominated by any other data points in P. Data point p_1 is said to dominate a data point p_2 if and only if p_1 is not worse than p_2 in any dimension and better than p_2 at least one dimension. In this paper, we define the higher value on each attribute, the better.

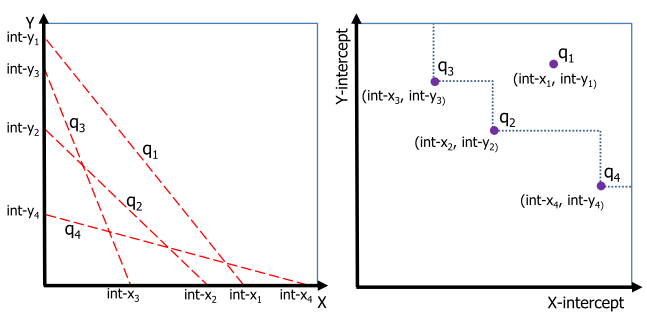
The skyline of local data objects of M_i (D_i) is denoted by $SK_i = skyline(D_i)$. In addition, we define a useful function which tests whether a given weighting vector and a given score c cross SK_i.

$$cross_{SK_i}(sf, c) = \begin{cases} true & \exists d\,(d \in SK_i \wedge sf(d) \geq c) \\ false & otherwise \end{cases}$$

3.6 A Set of Dominating Queries

The query list (QL) at BS must be able to handle a large number of queries. However, the affected area of some queries are possibly fully contained in the affected area of another set of queries called a set of dominating queries (DQ). To identify DQ, we solve the axis intercepts of the linear equations (the scoring functions with their thresholds in Fig.3a), then those axis intercepts are plotted in the new space called *the intercept space* as shown Fig.3b. We define DQ as the queries lying on the skyline (the lower the better) in the intercept space. Therefore, preserving the integrity of DQ ($\{q_2, q_3, q_4\}$) also ensures the integrity of top-k answers for remaining queries $QL \backslash DQ$ ($\{q_1\}$) in the query list.

Definition 2: A set of dominating queries DQ contains all the queries that belong to the skyline in the intercept space.



(a) The threshold lines (b) The intercept space

Fig. 3 Example of representing 4 queries

Theorem 1: If q_1 dominates q_2 in the intercept space, then $TOP_{q_2} \subseteq TOP_{q_1}$.

Proof: Assume there exists $d \in TOP_{q_2}$ but $d \notin TOP_{q_1}$. This means d falls in the affected area of q_2 but not q_1. Since q_1 dominates q_2 in the intercept space, the affected area of q_2 is fully contained in the affected area of q_1. This leads the contradiction.

Theorem 2: Any $q \notin DQ$, there exists at least one query dominating q in the intercept space.

Proof: Assume that $q \notin DQ$ and there is no query dominating q in the intercept space. This cannot happen because by Definitions 1 and 2, q must belong to DQ.

3.7 Answering Top-k Queries Using Views

BS preserves entire active previously posed continuous queries with their valid answers. In the presence of this knowledge, we can use that to determine whether the data objects in DP are sufficient to satisfy the new coming query. By the concept of answering top-k queries using views in [9], [11], given a set of views and a top-k query, these algorithms solve the linear programming optimization on a convex region of the set of views to find the minimum score of the top-k query that can be guaranteed by these views. However, the set of the views (previously posed queries) to be used in the execution in our proposed method are DQ ($DQ \subseteq QL$). It is noted that a new query q which does not belong to DQ can be definitely answered by using the views of the dominating queries that dominates q, but the converse is not always true. Due to the limitation of the space, the algorithm details can be found in [9], [11].

4. Proposed Algorithms

4.1 Outline of the Proposed Method

There are many local nodes which are holding different sets of data objects in the system. Without any technique, for each single query, BS cannot help asking every node to send back all possible final answers back to BS. To prevent that situation, given a first query q_1, BS requests the score summary information by using small-sized histograms to roughly decide the score threshold. This may take multiple iterations to be finished. BS uses the estimated threshold to request only a small number of data objects whose scores surpass the given threshold back to BS. Apart from cost of transferring desired data objects to BS, the number of messages also consumes a large cost of communication especially when flooding query messages to irrelevant nodes. Together with the given threshold, all local nodes' skylines aggregated at BS in the initialization, called skyline indexes, can identify a small number of nodes which are involved with query processing while ensuring the completeness of final answers.

Given existing queries q_1, q_2 and another new coming query q_n, it is possible that BS can immediately answer q_n since BS already has some data objects retrieved so far from q_1 and q_2. To check this, it can be categorized into 3 cases as follows:

- The answer space of q_n is covered by either q_1 or q_2 (DOM). This happens when q_n is not included in a set of dominating queries. Therefore, final answers of either q_1 or q_2 can sufficiently answer q_n.

- The answer space of q_n is covered by together q_1 and q_2 (VIEWS). This means the final answers of some materialized views (q_1 and q_2) can sufficiently answer q_n. This can be identified by the method discussed in Sect. 3.7. This kind of queries is included in a set of dominating queries.

- Otherwise (HIST), existing data objects at BS are not theoretically guaranteed to be enough to answer q_n. Here, q_n has to be executed by estimating the threshold as same as the first query (q_1). This is the most expensive case. However, due to existence of some potential data objects at BS, more appropriate range of histograms can be decided resulting that it takes very few iterations to be finished.

The local nodes must be able to monitor the changes of data and identify which one is involved or affects the final answers of active queries in QL. For this aim, our method disseminates a top-k subscription S_q, which contains the information about the affected area of q, to local nodes from BS, and local nodes keep all received subscriptions in its own subscription list (SL). Local nodes have the responsibility to keep an eye on local data objects matched with the subscription. If an existing data object or an incoming update matched with the subscription but it has never been sent to BS so far, this data object must be sent to BS because it potentially affects the final answers of the queries in QL. The benefit is that a local node monitors only a small affected area from an entire data space, so a large number of unnecessary data objects and data updates can be pruned. In practice, a data request, which BS asks local nodes to send the data objects to BS for the first time, is also proceeded via subscriptions.

It can be expensive to construct the subscriptions of all active queries in QL and to flood those subscriptions to all local nodes ($|QL| * N$ messages). Therefore, we bind the top-k subscription if and only if that query is in DQ. Because of skyline indexes, BS uses them together with the threshold line from S_q to identify the nodes that do not certainly contribute any final answers by using function $cross_{SK_i}(q.sf, q.ths)$. These subscriptions will be bound to a small and different group of local nodes. As a result, BS can save cost by avoiding making any requests to those irrelevant nodes.

In conclusion, for a new query q_n, in the case of HIST, it must pay communication cost for both threshold estimation and issuing subscriptions. In the case of VIEWS, BS needs to pay only the cost of issuing subscriptions, while DOM requires nothing.

Algorithm 1 Base station's procedures

A new query $qk = (sf, k)$ **inserted at BS**
1: $ths \leftarrow k$-th highest score of $qk.sf(DP)$
2: $q \leftarrow (sf, k, ths)$
3: **if** q is not a dominating query **then**
4: QL.put(q)
5: **else if** q can be answered by using views **then**
6: $BindSubscription(q)$ // Algorithm 4
7: QL.put(q')
8: Update DQ
9: **else**
10: $score_{min} \leftarrow q.ths$
11: $score_{max} \leftarrow max\left\{q.sf(d_i) | d_i \in \bigcup_{i=1}^{N} SK_i\right\}$
12: $q \leftarrow ThsEst(q, score_{min}, score_{max})$ // Algorithm 3
13: $BindSubscription(q)$ // Algorithm 4
14: QL.put(q')
15: Update DQ

An old query $q = (sf, k, ths)$ **deleted from BS**
16: QL.remove(q) and announce to nodes keeping q
17: **if** $q \in DQ$ **then**
18: Inform nodes to invalidate S_q
19: Update DQ
20: Bind some additional subscriptions if necessary

BS receives data update d'**'s information**
21: Add or remove d' from DP
22: Update answers and thresholds of affected queries
23: Update DQ
24: Bind some additional subscriptions if necessary

BS receives skyline updates from M_i
25: Invalidate the existing skyline of M_i which is dominated by this update
26: Store new skyline tuples

Algorithm 2 Threshold Estimation (ThsEst)

Histogram request at BS
Input: Query $q = (sf, k, ths)$, $score_{min}$, $score_{max}$
Output: Query $q = (sf, k, ths)$ // threshold ths changed
1: $n_b \leftarrow \left\lceil \sqrt{q.k} \right\rceil$
2: $r \leftarrow (\frac{score_{max} - score_{min}}{n_b})$
3: **repeat**
4: $pths = score_{min}$
5: Construct equi-width histograms $H\left\langle H_1, H_2, \ldots, H_{n_b} \right\rangle$ on the range $[score_{min}, score_{max}]$
6: Histograms request on M_i if $cross_{SK_i}(q.sf, score_{min})$
7: $H \leftarrow$ Combine the histograms from nodes
8: $CF \leftarrow 0$
9: **for** $i = n_b \rightarrow 0$ **do**
10: $CF \leftarrow CF + H_i$
11: **if** $CF \geq q.k$ **then**
12: $score_{min} \leftarrow max(score_{min}, score_{max} - ir)$
13: **until** $score_{min} \leq pths \vee CF \leq (1 + \gamma)q.k$
14: $q.ths \leftarrow score_{min}$

Histograms request at node M_i
Input: Query q, Histograms H, $score_{min}$, $score_{max}$
Output: $H\left\langle H_1, H_2, \ldots, H_{n_b} \right\rangle$
15: **for all** $d_j \in D_i$ **do**
16: **if** $score_{min} \leq q.sf(d_j) \leq score_{max}$ **then**
17: $p = \left\lfloor \frac{(q.sf(d_j) - score_{min}) \cdot q.k}{score_{max} - score_{min}} \right\rfloor$
18: $H_p \leftarrow H_p + 1$

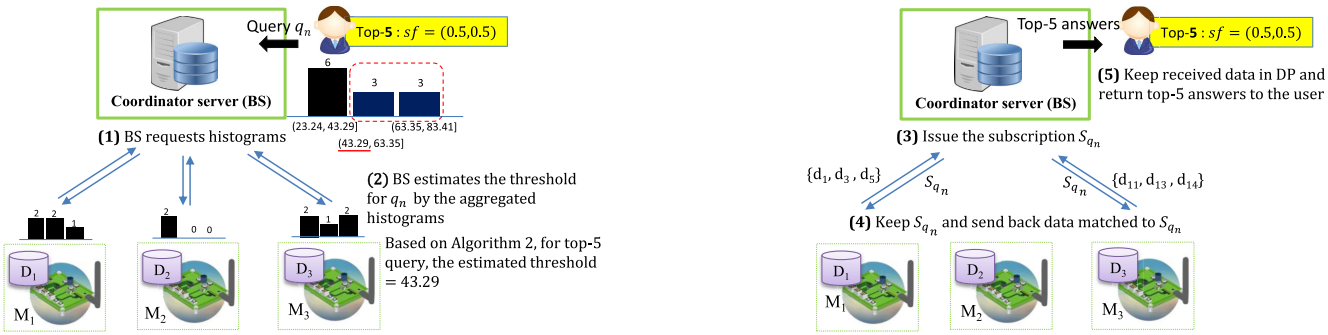
4.2 Initialization

The initialization phase is done once at every node. In order for BS to be able to decide effective query requesting, each node has to contribute some information to BS. That is the associated data tuples of the skyline of the local data objects called skyline indexes (SK_i). It is noted that exchanged data are only associated data tuples (numerical attributes), not data objects, which are smaller in size. When BS receives these skyline indexes from the local nodes, it constructs a table to store this information.

4.3 Handling a New Query

The algorithm of this phase is shown in Algorithm 1, lines 1-15. When there is a new query injected by a user at BS, firstly the initial threshold of the query is set as the k-th score of valid data objects in DP corresponding to the query's scoring function $q.sf$. Afterward, our method preliminarily analyzes the query with its initial threshold. In the case that q is included in DQ, our method simply stores this query in QL, because this query can be answered from the dominating query which dominates it, and the initial threshold is actual. In the case that q is not included in DQ, our method later examines whether q is answerable by a combination of views. If answerable, BS does not require to retrieve further data objects, because all data objects in DP are sufficient and the initial threshold is also actual. Besides, this query becomes a new dominating query, so DQ is needed to be updated, and a subscription of a new dominating query in DQ must be constructed and bound (Algorithm 3). If unanswerable, this means the initial threshold is not actual, and the data objects in DP is insufficient to answer the query. To construct the top-k subscription and request the data objects by using this initial threshold can incur a lot of unnecessary transferred data objects, since this initial threshold is possibly far from the actual one.

To solve this problem, BS calls the procedure to estimate the threshold by using 1-dimensional equi-width histograms in Algorithm 2, lines 1-14. We choose the number of bins of histograms equal to $\left\lceil \sqrt{q.k} \right\rceil$ because this parameter should depend on k but must be smaller than k. Because there is no best number of bins for every data distribution, without the knowledge of the real data distribution, a good rule of thumb is to use the square-root choice for the number of bins. BS knows the skyline points of all nodes, all possible top-1 answers, so the maximum score ($score_{max}$) can be definitely known by selecting the greatest score in skyline indexes. The minimum score of histograms ($score_{min}$) is a known and latest estimated threshold. Initially, $score_{min}$ is set as equal to the initial threshold $q.ths$. BS sends a request the histograms from some relevant nodes M_i where $cross_{SK_i}(q.sf, score_{min})$ returns true (line 6). Each requested local node constructs histograms by counting the frequencies of local data objects' scores on each bin (denoted by a score range of a bin). As a result, the merged histograms H at BS (line 7) represent the frequency of the global scores of each score range. We count cumulative frequencies (CF) from the bin which has the highest score range to lower ones, and we stop at the bin that gives



(a) Threshold estimation

(b) Issuing the the subscription and retrieving data objects

Fig. 4 Overview procedures for a new query q_n

Algorithm 3 Bind Subscription

On bind subscription of q at BS
1: Construct subscription S_q
2: Bind S_q to M_i if $cross_{SK_i}(S_q.sf, S_q.ths)$
On receive subscription S_q at M_i
3: Store S_q in SL_i
4: Send $d_j \in D_i$ which $S_q.sf(d_j) \geq S_q.ths$ and is not yet sent to BS

Algorithm 4 Maintenance procedure at M_i

Data object insertion of d'
1: $D_i \leftarrow D_i \cup \{d'\}$
2: **if** d' is not dominated by SK_i **then**
3: Invalidate data points which d' dominates in SK_i
4: $SK_i \leftarrow SK_i \cup \{d'\}$
5: Inform BS a data tuple d'
6: **if** d' is included in SL_i and not sent to BS yet **then**
7: Send d' to BS and mark d' as already sent to BS
Data object deletion of d'
8: $D_i \leftarrow D_i \backslash \{d'\}$
9: **if** $d' \in SK_i$ **then**
10: Execute constrained skyline to update SK_i
11: Send new skyline candidates to BS
12: **if** d' was already sent to BS **then**
13: Inform BS about data deletion

$CF \geq q.k$. We set new $score_{min}$, i.e., the latest threshold as the lower bound score of that bin. In the case that CF is still high, using this threshold to request final answers can incur many unnecessary data objects to be retrieved, so we repeat requesting histograms again by using new $score_{min}$ until it meets the stopping criteria as follows;

1. When the new estimated $score_{min}$ is not different from the previous one,
2. When the cumulative frequencies are less than $(1 + \gamma)q.k$ where we allow some $\gamma q.k$ false positive data objects to be transferred, and
3. When it exceeds the maximum limit.

Note that this algorithm can incur some false positives but not incur false negatives, so the correct final top-k answers can definitely be returned to users.

In each iteration, $score_{min}$ gradually converges to the actual threshold while the number of nodes to be disturbed is likely to decrease. The initial threshold could be quite far from the actual one. However, in latter queries when some potential final answers of the previously posed queries are available, the initial threshold is close to the actual threshold, so the threshold estimation process is finished within one or a few iterations.

4.4 Issuing the Subscription and Acquire Top-k Answers for a Query

BS constructs a top-k subscription if it is a dominating query (Algorithm 1, lines 6-8 and 13-15). The subscription contains the query's scoring function ($q.sf$) and the query's latest threshold ($q.ths$). To prevent flooding, BS selectively binds the subscription to nodes whose skyline indexes overlap with the latest threshold ($cross_{SK_i}(S_q.sf, S_q.ths)$). Therefore, only a subset of nodes receives this subscription, i.e., each node receives a different set of subscriptions based on the necessity. Especially, if the threshold is equal to the actual threshold, this subscription will be bound to only a minimum number of needed nodes. Then, the nodes store the received subscription as well as send the local data objects that are matched with the subscription and not yet sent so far to BS. The data objects sent to BS must be marked as already sent to BS to avoid redundant transferring. This procedure is shown in Algorithm 3.

4.5 Top-k Answers at the Base Station

The top-k answers for a query q which is in QL are simply the first k-ranked data objects among all data objects in DP according to the scoring function $q.sf$. This approach can be enhanced by an implementing index of reverse top-k queries [18], [23], which is beyond the scope of this paper.

(Running Example)

Figure 4 shows a running example when posing a new query which is the case of HIST. In Fig.4a, after a user poses a top-5 query to BS, BS requests and aggregates histograms from distributed nodes by using $\#bin = 3$. Then, BS estimates the threshold by using those aggregated histograms. According to the histograms, we stop at the second bin

$CF = 6$ resulting in $score_{min} = 43.29$. Assuming $\gamma = 0.5$, $CF < (1 + 0.5) \cdot 5$, so BS does not need to repeat requesting histograms. Hence, in Fig.4b, BS constructs and issues the subscription by using 43.29 as a threshold. Due to the skyline indexes, BS may not need to issue the subscription to all nodes, i.e., only M_1 and M_3 are requested in the example. After that, nodes keep the received subscription and send back data objects that are matched with the subscription to BS (6 objects returned). Those data objects sent back from nodes and the available data objects in DP are sufficient to correctly answer the top-k query. Finally, BS returns the final answers to the user.

4.6 Maintenance on Query Updates and Data Updates

A query can be arbitrarily injected or withdrawn from BS while data objects held by each node can dynamically change as time passes. The following procedures are performed in such a case.

(Query injection) A new query is handled as the same procedure as stated in Sect. 4.3.

(Query withdrawal) A query can be simply removed from QL. In the case of $q \in DQ$, DQ is updated and it is possible that some queries in QL become new dominating queries. Therefore, BS need to construct and disseminate the subscriptions of those new dominating queries. After that BS has to inform the nodes which store this query's subscription to invalidate it from SL_i as shown in Algorithm 1 (lines 16-20).

(Data insertion) A new inserted data item is possibly included in the final answers of some queries and affects the change of local skyline. In the case that a new data object is not dominated by SK_i, this means that the new data item becomes a part of new SK_i. Then, the node sends the associated data tuple of the new data object to BS, and BS updates or issues subscriptions as necessary (Algorithm 1 lines 21-24). Both BS and local nodes do the same procedure of updating skyline indexes. They simply include the new data object in SK_i and invalidate data objects dominated by this new data object (Algorithm 1, lines 25-26). In the case that the new inserted data object is matched with one of the stored subscriptions in SL. This data object must be sent to BS and is marked as it was already sent to BS. Otherwise, the node keeps the new data object without sending to any node. Since most cases should fall in the latter case, the large amount of sending unnecessary data objects can be reduced. This procedure is shown in Algorithm 4 (lines 1-7).

(Data deletion) Data deletion at a node also has a same effect as data insertion. In the case that a deleted data item is in SK_i, our method performs a similar procedure as data insertion. The node has to send new skyline indexes candidates to BS as well as inform about skyline updates due to the data deletion. The new candidates can be identified by executing constrained skyline processing on the area which only a data object to be deleted is dominating. Finally, if this deleted data object was already sent to BS before, then this node must inform BS to invalidate it. Due to the disappearance of the data object, some queries in QL are possibly affected, so those affected queries must be re-calculated their new threshold.

(Data modification) Fundamentally, data modification can be taken into 2 basic operations including data deletion of old values and data insertion of new values. These two basic operations make the communication with BS based on the necessity. Hence, a data modification can be processed and pruned locally if (1) an old data tuple is not in the current skyline and was not already sent to BS (2) a new data tuple does not become a new skyline point and is not matched with any subscriptions. In the case that either of these conditions is not satisfied, a data modification can be equally translated to either data insertion or data deletion.

In addition, we can further cut some communication cost and latency if both conditions are not satisfied by informing BS of a data modification (old values and new values) in one round of communication. Then BS modifies the values in line 21, Algorithm 1 instead of executing either insertion or deletion. This can help BS call lines 21-24 in Algorithm 1 only one time.

In Sect. 5, we focus only 2 basic operations including data insertion and data deletion without using a dedicated maintenance for data modification.

4.7 Algorithm Correctness

The algorithm for estimating the threshold is designed based on the safety. It is noted that the possible maximum threshold cannot be higher than the k-ranked data object of the global dataset (D). Therefore, it is safe that there must be at least k data objects whose scores trespass the estimated threshold, and those data objects are the candidates to be included in the final top-k answers.

We claim the correctness of the method proposed as follows; Assume that at one period of execution time, a data object d' which originally belongs to node M' is included in the final answers of the query q, but this data object d' has not been sent to BS. This causes incorrect final answers. Therefore we proof by contradiction to claim this will not happen by our method. (1) If M' has been received a subscription of q, d' must be sent to BS because the score of d' ($q.sf(d')$) must be higher than the threshold unless there are at least other k data objects which contribute higher scores than d' that means d' is not included in top-k final answers. This makes the contradiction; therefore, d' must be sent to BS. (2) If M' has not been received a subscription of q, this means all local data objects of M' and its SK have lower scores than the threshold. Hence, a local data item d' definitely has a lower score than the threshold. This leads to the contradiction. As a result, $q.sf(d')$ must be higher than the threshold because we firstly assume that d' is one of the final answers.

4.8 The Procedure When a Node Joins or Leaves

Even though we assume that the topology is unchanged, in the case that a node M_i joins the system, the node must send its SK_i to BS. BS then issues a set of necessary subscriptions based on the received SK_i. After that M_i must response to those subscriptions by sending the data objects matched with those subscriptions to BS. In the case that a node M_i leaves, BS must invalidate the stored SK_i and the valid data objects of M_i in DP. This can be handled as a bulk of data deletions. These events trigger the procedure for data insertion and data deletion as explained in Sect. 4.6.

5. Performance Evaluation

The experiments were conducted by using an event-based simulator implemented in Java. We assumed that the coordinator server (BS) can directly communicate with all nodes. The range and default setting of these parameters are expressed in Table 1. Each node M_i initially holds $|D_i|_0$ data objects and each data object is large $Size_{obj}$ bytes. Firstly we initially injected $|QL|_0$ queries into the system. The weighting of each dimension of a query (w_i) is uniformly random and normalized with $\sum_{i=1}^{i=m} w_i$ resulting $\sum_{i=1}^{i=m} w_i = 1$. Our proposed method can support any arbitrary k for each query, but for fair comparison with other methods, value k of each query is a uniformly random integer between 1 to k_{max}. Afterward, we simulate the dynamic changes occurring in the system as events including data insertion/deletion and query insertion/deletion. For a fair comparison, we conduct the experiments by using 3 synthetic datasets covering most of experimental settings in [14] and a real dataset as follows;

1. **Clustered dataset (CL):** Each node randomly draws a coordinate of its centroid of a cluster on the m-dimensional data space. The data value on each dimension is independently generated by Gaussian distribution with a specified variance. Hence, each node will hold a group of data objects that close to each other but different from other nodes.
2. **Uniform dataset (UN):** The data value on each dimension is uniformly pointed in the data space.
3. **Anti-correlated dataset (AN):** This dataset is generated referring to [19] and is equally partitioned to nodes. The characteristic of this dataset is that a data object tends to have high value on one attribute and low on the others. Most of methods suffer due to the large number of possible top-k answer candidates, such as skyline (1-skyband) and K-skyband.
4. **Real dataset (RL):** This dataset includes the product information and user reviews from amazon.com.

Table 1 Simulation parameters

Parameter	Default	Range
Number of nodes N	500	100-1000
$Size_{obj}$ [byte]	300	100-1000
$\|QL\|_0$	1000	100-10000
$\|D_i\|_0$	150	100-500
k_{max}	15	1-60
Dimensionality m	3	2-8
Datasets	CL	CL, AN, UN

5.1 Results of Communication Cost

We fixed the number of events at 200,000 per experiment. This should be high enough to observe behaviors due to the effect of dynamicity in long-term. Since a user hopes to monitor top-k answers for a long period of time, the dynamicity of data is more likely to occur more often than the query dynamicity. Therefore, we set the chance of occurrence for each event as follows; data insertion/data deletion at 49% each and query injection/query withdrawal at 1% each. We varied parameters and datasets in each experiment to study the effects of these parameters. Due to the lack of space, we omit graphs of some datasets which share the same behaviors and characteristics.

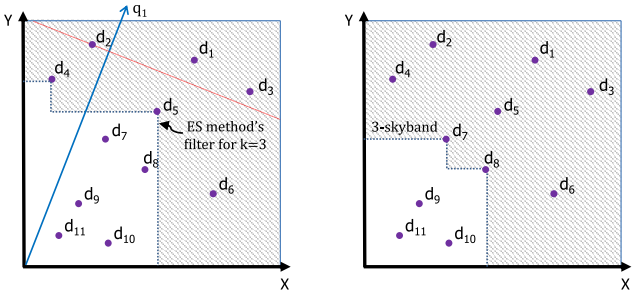
5.1.1 Measurement

We recorded the final cost of communication defined by volume of transferred data in the system as a metric to compare the performance. The cost of communication was counted by how many data objects and queries being necessary to transmit between BS and local nodes. We defined the size of floating point and integer equal to 8 bytes and 4 bytes respectively. Therefore, for example, a top-k query attached with the threshold consisting of a scoring function (m floating points), value k (1 integer) and a threshold (1 floating point) is $(8(m + 1) + 4)$ bytes.

5.1.2 Benchmarks

We implemented the following methods for comparing with our proposed method.

1. **Centralized method (CEN):** The baseline which all data objects and information of updates are sent to BS regardless of queries.
2. ***Enhanced scheme* proposed in [14] (ES):** This method produces a top-k point-based filter by utilizing dominant properties to guarantee that any data points which are discarded by this filter are impossible to be in final top-k answers. The filter is unavoidably issued to every nodes. However, due to data updates, the filter may become invalid when using for a while that leads to renewal of the filter by re-issuing a new filter to entire nodes. According to the filter, local nodes can prevent sending irrelevant data objects in some degree. This method was proposed for WSNs in which sensor data with specified expiration time are relayed via intermediate nodes.
   We re-implemented this method with the same concept to use in our assumed topology to compare the performance with our proposed method. Our assumed system

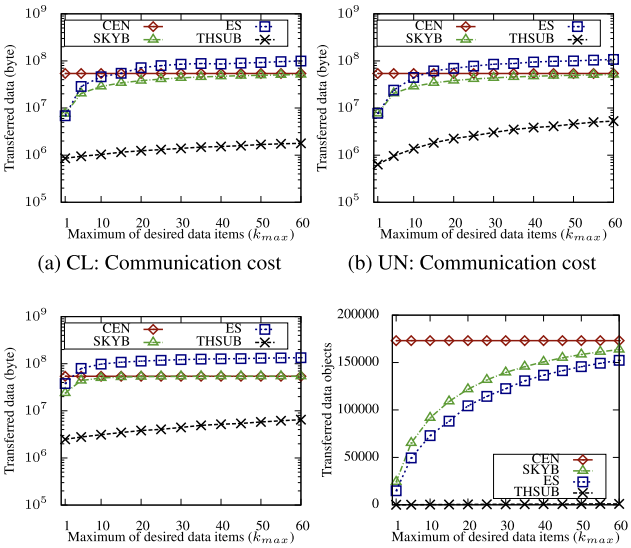


(a) ES method (b) K-skyband

Fig. 5 The affected area of 3 methods on a top-3 query

has no data expiration time but arbitrary data deletion, so local nodes need to send a message to inform about disappearance of a data object if the deleted object is sent to BS so far. This method only takes a single parameter into account. This parameter defines the maximum value of k in top-k queries that it can answer, so we assigned this parameter as k_{max} in our experiments.
In Fig.5a, the point filter for $k = k_{max} = 3$ includes d_4 and d_5. Using this filter, $\{d_1, d_2, d_3, d_4, d_5, d_6\}$ will be returned to BS. If data insertion or deletion occurred in the gray-shaded area (the affected area of the filter), nodes must report those data updates to BS. Otherwise, the updates can be neglected. It is noted that, given the weighting vector of q_1 as shown, the top-3 answers of q_1 only includes d_1, d_2 and d_3.

3. **K-skyband (SKYB):** The data objects which belong to K-skyband are sufficiently enough for answering any top-k queries where $k \leq K$. BS simply aggregates K-skyband from every local node, so any top-k queries at BS can be answered intermediately by the aggregated data objects. Due to the data dynamicity, each node continuously maintains the K-skyband and sends a new data update to BS when that new data update belongs to its local K-skyband. It is noted that the size of K-skyband is bigger than the real necessity, and in high dimensionality and high value K, K-skyband of node M_i possibly includes all data objects D_i, i.e., transferring all data objects like the CEN method. In the experiments, we assign K in K-skyband equal to k_{max}.
In Fig.5b, the data objects that belong to 3-skyband consists of $\{d_1, d_2, \ldots, d_8\}$. The gray-shaded area exhibits the affected area of the 3-skyband. K-skyband is calculated locally for each node unlike the ES method which is the global point-based filter issued from BS. Therefore, there is no communication cost for filter construction. Only data insertion or deletion occurred in the gray-shaded area (the affected area) must be reported to BS.
4. **Our proposed method (THSUB):** Our proposed method described in Sect. 4. Based on our parameter study, we set the maximum iteration for threshold estimation as 5, and $\gamma = 0.5$.



(a) CL: Communication cost (b) UN: Communication cost

(c) AN: Communication cost (d) CL: No.of sent objects

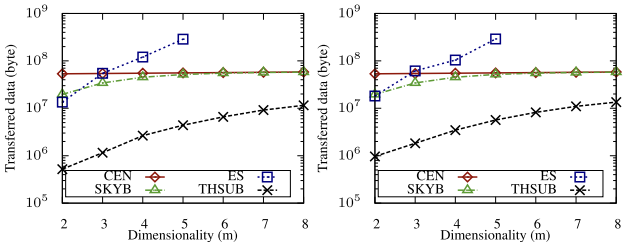
Fig. 6 Impacts of k_{max}

5.1.3 Impact of the Number of Desired Data Objects

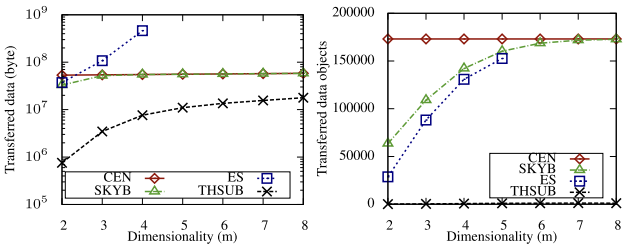
We first evaluate the performance on varying the number of desired data objects or value k defined in a query. The result of the CL, UN and AN datasets are shown in Fig.6a, Fig.6b and Fig.6c respectively.

According to the result for the CL dataset, for very small k_{max} (roughly from 1 to 15), the ES and SKYB methods perform better than the CEN method. This is because the set of top-k answer candidates and the affected area specified by the filter and K-skyband are still relatively small. A lot of updates occurred are not sent to BS. However, as k_{max} increases ($k_{max} \geq 15$), the ES method's cost becomes over the CEN method; because, apart from sending data objects, the cost of filter update is also expensive. For high k_{max}, the setup filter becomes stale more rapidly than smaller k_{max}. As k_{max} increases, the cost of the SKYB method converges to the cost of the CEN method because the size of K-skyband and entire local data objects are almost same and the affected area is large. Thus, most of updates are sent to BS. As a result, the total cost of the SKYB and CEN methods are similar. In contrast, the THSUB method has more efficiency since the cost linearly increases with k_{max} but lower than the comparative methods. This is because the top-k subscriptions are selectively diffused to some nodes. Due to the affected area more strictly controlled by the subscriptions, unnecessary updates are not reported to BS unlike the comparative methods which greedily aggregate all possible top-k answer candidates.

In the UN dataset, the THSUB method's cost grows slightly faster than the CL dataset. This is because, for uniform distribution, besides updates are more likely to match the subscriptions, the subscription update occurs more frequently. Therefore, the cost of the THSUB method grows



(a) CL: Communication cost (b) UN: Communication cost



(c) AN: Communication cost (d) CL: No.of sent objects

Fig. 7 Impacts of dimensionality m

up but significantly less than other comparative methods.

In the AN dataset, the SKYB, ES and THSUB methods suffer because the cost of initialization as well as the size of top-k candidates such as the number of skyline points, size of K-skyband and size of filters are object to be very large. The cost of the SKYB method is almost equal to the CEN method from $k_{max} = 15$, and the cost of the ES method surpasses the cost of the CEN method when $k_{max} = 5$.

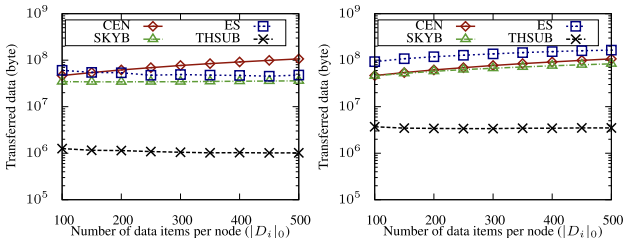
In Fig.6d, we show how many data objects are sent to *BS* which does not include skyline data tuples in the initialization. Our proposed method matches only final answers, therefore; only a few thousands of data objects are returned to *BS* unlike others.

5.1.4 Impact of Dimensionality

The results of the CL, UN and AN datasets are shown in Fig.7a, Fig.7b and Fig.7c respectively when varying the number of dimensions in data objects. We omit some results of the ES method since the amount of transferred data is significantly higher than others.

In [14], the authors have only reported the evaluation of the ES method on varying dimensionality by fixing k_{max} at 1, but our default setting sets k_{max} at 15. As the dimensionality increases, the cost of the ES method converges close to the CEN method because the size of K-skyband becomes large and almost same as sending entire local data objects of each node. The ES method significantly performs worse than the CEN method when $m \geq 3$ because the size of a filter to be set up at every node is large, so the cost of filter setup becomes very expensive.

The THSUB method suffers when using the AN dataset especially due to huge skylines in the initialization phase and more frequent skyline updates sent to BS from each node. It is noted that the THSUB method performs worse



(a) CL: Communication cost (b) AN: Communication cost

Fig. 8 Impacts of $|D_i|_0$

as dimensionality increases; however, it is still superior than other competitive methods because the number of sent data objects is strictly controlled by subscriptions as shown in Fig.7d.

5.1.5 Impact of the Initial Number of Data Objects

We varied the initial number of data objects per node ($|D_i|_0$) in the network by keeping other parameters fixed. This increment influences the whole amount of data objects increasing. The effects of this factor in the CL and AN datasets are shown respectively in Fig.8a and Fig.8b.

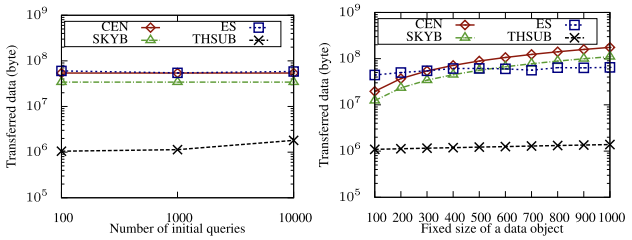
The result of the CL and UN dataset shares the same characteristics, using the CEN method, obviously the cost of data aggregation and the cost due to initialization grow linearly. The SKYB method remains almost constant in all settings because every local node has to send its own K-skyband to *BS* regardless of how dense of data objects in the global space. On the contrary, the high data density brings a positive effect on the ES and THSUB methods. When the data are dense, the affected area as well as the ES method's filters becomes smaller, so new updates that need to be sent to *BS* occur infrequently. In addition, the constructed filters become stale slower because the occurrences of new data updates that invalidate the old filters are expected to be lower in the ES method. Thus, the costs of the 2 methods slightly decrease as $|D_i|_0$ increases.

In the AN dataset, the affected area is covered by most of local data objects in the SKYB and ES methods. They have no gain over the CEN method. Furthermore, because of the cost of flooding new filters, the cost of the ES method goes beyond the CEN method.

Even though our proposed method pays higher cost in the AN dataset in comparison with the first two datasets, it does not have that negative effect from increasing $|D_i|_0$. This is because it subscribes only final top-k answers regardless of data distribution.

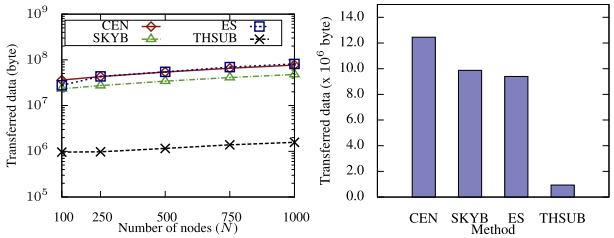
5.1.6 Impact of the Number of Initial Queries

Figure 9a shows the result of the CL dataset when varying the number of initial queries. Only the THSUB method is affected by the number of queries because the rest of comparative methods take only k_{max} into account regardless of the actual preference (scoring function) of each query. There-



(a) CL: Communication cost (b) CL: Communication cost

Fig. 9 Impacts of $|QL|_0$ and $Size_{obj}$



(a) CL: Communication cost (b) RL: Communication cost

Fig. 10 Impacts of N and a result of RL dataset

fore, the transferred data of those methods remain constant in every setting. Since the THSUB method considers every single query, it has good scalability that can support a large number of active queries at a time while consuming lower overhead than other methods.

5.1.7 Impact of Size of Data Objects

We assume that data objects which are sent to *BS* for answering top-k answers, have a fixed size. In this experiment, we varied the size of data objects in order to evaluate its effect shown in Fig.9b.

Our proposed method, instead of greedily aggregating data objects to *BS*, tries to carefully construct top-k subscriptions to request only final answers and inform the local nodes the precise affected area unlike the comparative methods. Practically, the actual final answers for top-k queries are a small portion of entire global data objects, and *BS* receives unnecessary data objects fewer than other methods. Therefore, the THSUB method gets less effects from varying the size of data objects, and a large amount of incurred cost is because of threshold estimation, skyline updates and subscription maintenance.

On the contrary, other methods receive and store many data objects at *BS* by query processing or data updates in order to guarantee the accuracy of the final answers. Even though, those data objects are not all included in the final answers of the active queries in *QL*.

5.1.8 Impact of the Number of Nodes

When increasing the number of nodes (N), the entire data objects in the system increases. Apart from our proposed method, the transferred cost grows linearly with parameter N because those methods aggregate the data objects from every node. For a large number of nodes, the THSUB method has to issue subscriptions to more nodes and more frequently, but it avoids flooding by selectively sending requests to only some of nodes. Therefore, while the transferred cost of the THSUB method also increases linearly, the growth rate is lower than the others. In the experimental results shown in Fig.10a, the THSUB method outperforms other methods.

5.1.9 Results on the Real Dataset

The real dataset from amazon.com is used for comparing the proposed method with the comparative methods. We selected products which contain complete information and have more than 15 user reviews (41,545 data objects). 3 numerical attributes were chosen and normalized; price (inverse value), average rating and useful comment ratio. We conducted the experiment by using the environment where most of default parameters are as same as the synthetic dataset except for the setting of $N = 100$ and the number of events at 50,000. Therefore, 15,000 data objects are equally divided to nodes, and the rest is for simulating data updates. In the result in Fig.10b, the SKYB and ES methods can save cost of communication compared to the CEN method. Still, the THSUB method outperforms other methods.

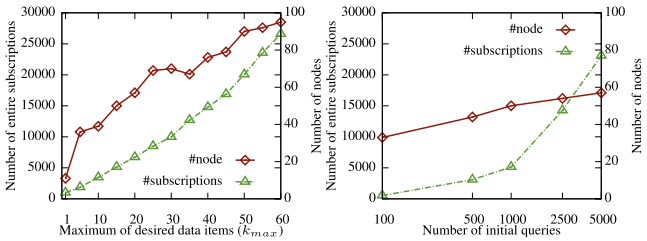
5.2 Analysis of the Proposed Method

Too many subscription messages can incur large communication cost. Without any techniques, in default setting, we need to send all subscriptions of 1000 queries to all 500 nodes which cost at least 500k subscriptions (not yet include subscription updates). Due to selectively forwarding only some subscriptions in *DQ* to some nodes by using skyline indexes, the proposed method can avoid sending numerous unnecessary subscriptions.

We conducted experiments by using the default setting as shown in Table 1 to simulate a snapshot of the system and measure the number of subscribed nodes and the number of transferred subscriptions. We set the number of events at 20,000 events including only data insertion and deletion with the same probability (no query dynamicity). In Fig.11a varying k_{max}, the total number of subscriptions is less than $30k$ subscriptions while the number of nodes having at least 1 subscription is less than 100 even $k_{max} = 60$. In Fig.11b varying the number of initial queries, it positively shows that increasing various queries into the system, the number of subscribed nodes is far less than the number of entire nodes as well as the total number of sent subscriptions.

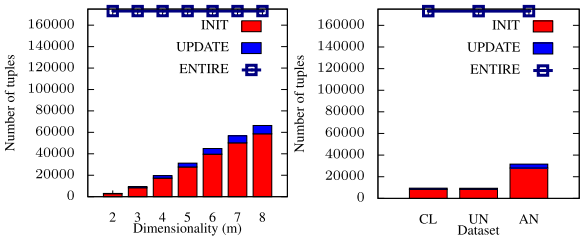
5.2.1 Skyline Indexes and Their Maintenance

We show the number of associated data tuples of local nodes' skylines due to the aggregation in initialization phase



(a) Against k_{max} (b) Against $|QL|_0$

Fig. 11 Results of #subscriptions and #subscribed nodes



(a) Against m (b) Against datasets

Fig. 12 The size of aggregated skyline tuples and updates

(INIT) and the updates (UPDATE) compared to the number of all transferred data objects in the network (ENTIRE). In Fig.12a, when increasing dimensionality, the size of skyline increases essentially as well as skyline updates. Furthermore, the number of skyline tuples largely increases in AN dataset as shown in Fig.12b. Even though, the cost of skyline aggregation is much expensive, but it is done once in the initialization after that we maintain them. The result also shows that the number of skyline updates (UPDATE) is small compared 20,000 data updates, but this differs on each kind of data distribution and dimensionality.

5.2.2 Iterations of Threshold Estimation

Threshold estimation takes multiple iterations to be finished, so this can cause a long latency. To prevent too many iterations of threshold estimation, a parameter γ is set to allow some false positives to be returned when retrieving final answers. However, it is not actually a wasteful transmission because these false positives once retrieved are possibly included in latter coming queries. From the experiment, $\gamma = 0$ takes the highest number of iterations and $0.5 \leq \gamma \leq 1.0$ provides the smallest number of iterations while the final number of transferred data objects are almost same. Therefore, we apply $\gamma = 0.5$ as default for all experiments.

Practically, in low dimensionality, threshold estimation is rarely invoked. From the experiments, only a first few queries take multiple rounds to finish because *BS* still does not have high potential data (cache in *DP* and top-k answers). Table 2 shows the total number of threshold estimations for the first 10 queries against dimensionality m. Averagely it takes less than 2 iterations per query, and the latter queries are expected to averagely take less than this or no communication latency. After that, the threshold estimation is finished within a single iteration.

Table 2 #threshold estimations in the first 10 queries

dimensionality	2	3	4	5	6	7	8
#Iteration	9	17	12	17	17	22	20

The actual latency largely depends on the real implementation. Regarding our default setting, the size of the payload for histograms is 56 bytes. If we assume that IEEE 802.15.4 which is a standard for low-cost and low-speed communication between devices is used for transmission, Fig. 5 in [24] shows that end-to-end transmission latency even up to 100 bytes of a payload is around 70-80 ms. For processing latency at the base station, as in our experiments on the default setting implemented in a commodity PC, this takes only 7-10 ms. To sum up, our proposed method incurs additional latency in the scale of milliseconds and should be suitable for most applications which the latency can be tolerated to this degree, for example, weather monitoring which periodically collects information from each node every 5 minutes. In contrast, in frequent update data model, the naive method which transfers massive data in real-time without reducing the volume possibly has other communication problems such as packet loss and bandwidth limitation.

6. Conclusion

In this paper, we addressed the issue of multidimensional continuous top-k query processing in distributed environments where data are horizontally partitioned. Our main aim is to reduce the communication cost of top-k query processing between the coordinator server where users pose top-k queries and distributed nodes where data objects are dynamically inserted or deleted.

Our propose method lets the coordinator server inform nodes about the boundary of data space of user interests called a subscription. To prevent superfluous query requests, our method identifies a small set of needed subscriptions to notify local nodes. Together with pre-indexing of skyline information, not only the number of requests but also the number of nodes to be requested can be significantly reduced. Our approach also identifies whether previously posed queries and data objects requested so far can answer a new query. If so, that new query can be answered without communication overhead. Furthermore, we took into account dynamic data updates including insertion and deletion and proposed the maintenance mechanism that assures the completeness of delivered answers to users.

We showed the performance of our method through the simulation results on both synthetic and real datasets. The results explicitly indicate that our method is preferred to the compared methods in terms of the communication cost.

References

[1] V. Hristidis, N. Koudas, and Y. Papakonstantinou, "Prefer: A system for the efficient execution of multi-parametric ranked queries," SIGMOD, pp.259–270, 2001.

[2] D. Xin, J. Han, H. Cheng, and X. Li, "Answering top-k queries with multi-dimensional selections: The ranking cube approach," VLDB, pp.463–474, 2006.
[3] L. Zou and L. Chen, "Dominant graph: An efficient indexing structure to answer top-k queries," ICDE, pp.536–545, 2008.
[4] L. Zou and L. Chen, "Pareto-based dominant graph: An efficient indexing structure to answer top-k queries," IEEE Trans. Knowl. Data Eng., vol.23, no.5, pp.727–741, 2011.
[5] P. Cao and Z. Wang, "Efficient top-k query calculation in distributed networks," PODC, pp.206–215, 2004.
[6] S. Michel, P. Triantafillou, and G. Weikum, "Klee: a framework for distributed top-k query algorithms," VLDB, pp.637–648, 2005.
[7] H. Yu, H.G. Li, P. Wu, D. Agrawal, and A. El Abbadi, "Efficient processing of distributed top-k queries," DEXA, pp.65–74, 2005.
[8] W.T. Balke, W. Nejdl, W. Siberski, and U. Thaden, "Progressive distributed top-k retrieval in peer-to-peer networks," ICDE, pp.174–185, 2005.
[9] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering top-k queries using views," VLDB, pp.451–462, 2006.
[10] E. Baikousi and P. Vassiliadis, "Maintenance of top-k materialized views," Distrib. Parallel Databases, vol.27, no.2, pp.95–137, April 2010.
[11] N.H. Ryeng, A. Vlachou, C. Doulkeridis, and K. Nørvåg, "Efficient distributed top-k query processing with caching," DASFAA, pp.280–295, 2011.
[12] S. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," OSDI, pp.131–146, 2002.
[13] M. Wu, J. Jianliang Xu, X. Xueyan Tang, and W.C. Wang-Chien Lee, "Top-k monitoring in wireless sensor networks," IEEE Trans. Knowl. Data Eng., no.7, pp.962 –976, 2007.
[14] H. Jiang, J. Cheng, D. Wang, C. Wang, and G. Tan, "Continuous multi-dimensional top-k query processing in sensor networks," INFOCOM, pp.793–801, 2011.
[15] B. Babcock and C. Olston, "Distributed top-k monitoring," SIGMOD, pp.28–39, ACM, 2003.
[16] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of top-k queries over sliding windows," SIGMOD, pp.635–646, ACM, 2006.
[17] K. Pripužić, I. Podnar Žarko, and K. Aberer, "Top-k/w publish/subscribe: A publish/subscribe model for continuous top-k processing over data streams," Information Systems, 2012.
[18] A. Yu, P.K. Agarwal, and J. Yang, "Processing a large number of continuous preference top-k queries," SIGMOD, pp.397–408, ACM, 2012.
[19] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," ICDE, pp.421–430, 2001.
[20] Y.Y. Xiao and Y.G. Chen, "Efficient distributed skyline queries for mobile applications," J. Computer Science and Technology, vol.25, no.3, pp.523–536, 2010.
[21] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis, "On efficient top-k query processing in highly distributed environments," SIGMOD, 2008.
[22] A. Vlachou, C. Doulkeridis, and K. Nørvåg, "Distributed top-k query processing by exploiting skyline summaries," Distrib. Parallel Databases, vol.30, no.3-4, pp.239–271, Aug. 2012.
[23] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, "Reverse top-k queries," ICDE, pp.365 –376, 2010.
[24] X. Liang and I. Balasingham, "Performance analysis of the ieee 802.15. 4 based ecg monitoring network," Proc. 7th IASTED International Conferences on Wireless and Optical Communications, pp.99–104, 2007.

Kamalas Udomlamlert received the B.E. degree in Computer Engineering from Kasetsart University, Bangkok, Thailand, in 2011 and received the M.Sc degree in Information Science and Technology from Osaka University, Osaka, Japan in 2013. Currently, he is pursuing a Ph.D at Osaka University. His research interests include distributed databases and information retrieval.

Takahiro Hara received the B.E., M.E, and Dr.E. degrees in Information Systems Engineering from Osaka University, Osaka, Japan, in 1995, 1997, and 2000, respectively. Currently, he is an Associate Professor of the Department of Multimedia Engineering, Osaka University. His research interests include distributed databases, peer-to-peer systems, mobile networks. He is a distinguished scientist of ACM, a senior member of IEEE, and a member of three other learned societies.

Shojiro Nishio received his B.E., M.E., and Ph.D. degrees from Kyoto University in Japan, in 1975, 1977, and 1980, respectively. He has been a full professor at Osaka University since August 1992. Dr. Nishio has received numerous awards during his research career, including the Medal with Purple Ribbon from the Japanese Emperor in 2011, which is awarded to people who have made outstanding and important contributions in academic fields, arts and sports. He is also a fellow of IEEE, IEICE and IPSJ, and is a member of five learned societies, including ACM.