M-027

水平分割データに対する連続 Top-k 問合せ処理

Continuous top-k query processing on horizontally-distributed data

Kamalas UDOMLAMLERT† Takahiro HARA† Shojiro NISHIO†

1. Introduction

In large databases, top-k query processing provides the capability of delivering *k* most preferable data to end-users. Ranking results depend on the data type and preferences of users. In this research, we focus on multi-numerical attribute databases where data are ranked by specified scoring functions from users. This problem becomes more challenging when a database is horizontally-partitioned among many nodes in distributed systems. In other words, a node holds a partial set of the entire database, and those data as well as queries are dynamically updated e.g. insertion and deletion.

In this situation, the coordinator server (CO) which takes a role of processing answers for end-users has no clues about the global ranking. Without any technique, it cannot help retrieving back all data from every local node to ensure that they are correctly ranked. An example is distributed online stores where each store contains information of products, and users want to monitor the *k*-best available products in real time (continuous queries). It is noted that, when a new product available but not in the top-*k* rank, sending it to a user is considered wasteful transmission.

In [1], the authors proposed that retrieving only data in K-skyband is sufficiently enough for answering all possible top-*k* queries where $k \leq K$. The method proposed in [2] makes use of dominant graphs for aggregating all possible candidates like [1] and also constructs the filter to be setup at every node to prevent unnecessary updates. However, an obvious weak point of both methods is to greedily aggregate answers regardless of actual preferences of users. Therefore, these methods may retrieve data over the necessity. This incurs the scalability issue in terms of data transferring cost between the coordinator and local nodes.

This paper proposes an efficient distributed continuous top-*k* query processing method for the situation explained above by aiming at 3 aspects

1. Avoiding flooding scheme that disturbs all nodes independent of whether those nodes contribute the final answers, by deriving benefits from pre-initialized local nodes' skyline
2. Utilizing the answers of previously-posed queries so far to benefit latter incoming queries
3. Reactively sending only necessary updates

2. Preliminaries

2.1 System environment

The distributed network consists of N nodes including a centralized coordinator (CO) and N-1 local nodes (M_1, ..., M_{N-1}) Every local node has a logical connection to CO. CO takes a role of processing queries from users while entire database is horizontally distributed among nodes. A data object has a fixed size associated with its identifier (*m*-dimensional numeric attributes) represented as a data tuple $d_j = (d_j[1], \ldots, d_j[m])$. Data objects are arbitrarily inserted or deleted at nodes as time passes.

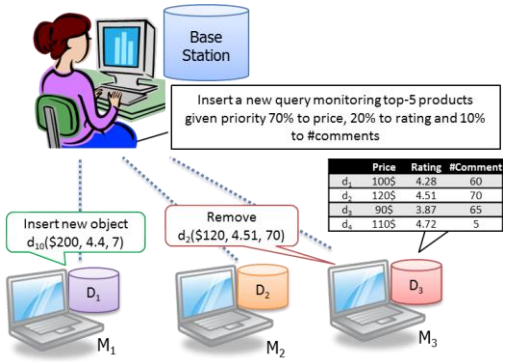

Figure 1 Example Scenario

2.2 Continuous top-k query

A top-k query $q = (sf, k)$ from a user is defined by a linear scoring function $sf = (w_1, \ldots, w_m)$ and the number of desired items k. The score of a data object w.r.t. query q is calculated by $sf(d_i) = \sum_{j=1}^{j=m} w_j \cdot d_i[j]$ where w_j represents weighting at j[th] dimension. $TOPq = \{d_1, \ldots, d_k\}$ stands for the set of answers for query q. In fact, in the Euclidean space, top-*k* and non-top-*k* answers can be easily divided by giving the score of *k*-th ranked answers $sf(d_k)$ as a threshold. The example is shown in Fig.2.

A continuous query is different from a snapshot query that a user wants to continuously monitor on the latest final answers. Therefore, our proposed framework assumes two components at CO including the query list (QL) and the data pool (DP). QL records current active queries, and DP stores valid data objects retrieved so far. For this aim, the system must ensure that data objects in DP satisfy all active queries in QL all the time.

2.3 Top-*k* subscription

The top-*k* subscription $S_q = (sf, ths)$ of query q is denoted by a scoring function and a threshold. This is disseminated to local nodes to let them know which data objects or data updates must be reactively sent back to CO. However, it is difficult to determine the threshold because CO does not know the global data distribution. We use the threshold estimation by using histograms to define the threshold. This threshold must satisfy that the number of data objects that are higher than threshold is not less than the parameter k defined in the query $|\{d_i | d_i \in D \wedge S_q.sf(d_i) \geq S_q.ths\}| \geq |\{d_i | d_i \in DP \wedge q.sf(d_i) \geq q.ths\}| = q.k$. However, we do not need to construct the subscriptions for every query in QL and do not need to disseminate to every node. The procedure to identify the needful queries will be described in Section 2.5.

† Graduate School of Information Science and Technology, Osaka University

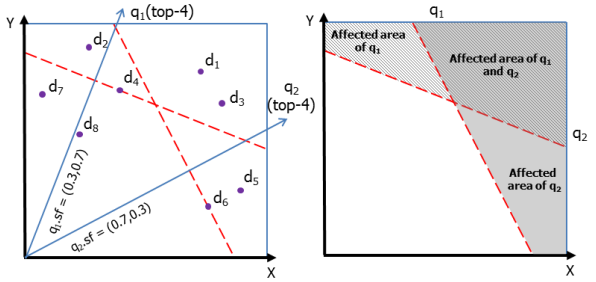

Figure 2: An example of 2 top-4 queries and their answers (left)
The affected area of those 2 queries (right)

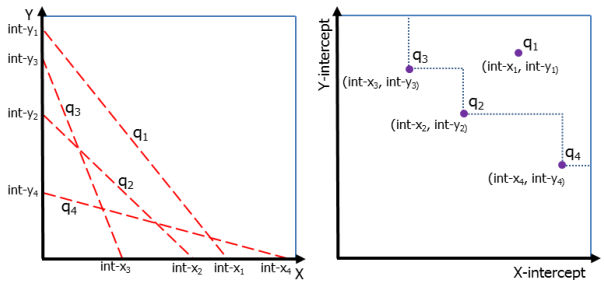

Figure 3: A set of 4 queries and their thresholds (left)
Their points in the intercept-space (right)

2.4 Skyline and its utilization

Skyline [3] is a type of operator which returns a set of data points P that for all points in P are not dominated by any other points. Data point p1 is said to dominate p2 if and only if p1 is not worse than p2 in any dimension and better than p2 at least one dimension. Inspired by the utilization of skyline in [4], nodes' skylines as data summarization enable the prevention of query flooding. We also make use of skyline in order to disseminate top-*k* subscriptions on only needful nodes to reduce the cost of communication because it is noted that, for a single top-*k* query, the number of involved nodes are smaller or equal to k ($k \ll N$).

2.5 A set of dominating queries

Among the queries in QL, it is possible that some queries are fully contained in some other queries (dominating queries). Therefore, if we ensure the completeness of dominating queries, the rest of the queries in QL can definitely be answered. To identify a set of dominating queries, we find the axis intercepts of linear equations of each query given by its scoring function and the *k*-th ranked score ($q.sf(d_k)$), then we plot those intercepts in the new space called *the intercept space* as shown in Fig.3. Among 4 queries, a set of dominating queries DQ is a set of queries that lie on the skyline including q_2, q_3 and q_4.

2.6 Answering top-k queries using views

In a snapshot of the system, there are currently active queries in QL and the complete answers of those queries (called views) are stored in DP. This raises a question whether they can be used for answering a new query. If yes, we do not need to pay extra cost to disseminate its top-*k* subscription or retrieve additional data. This can be solved by the linear optimization over the linear equations of queries in DQ. The details can be found in [5].

3. The procedures of our algorithm

3.1 Indexing skylines at the initialization

This is done once only in the initialization of the system. CO, to be able to efficiently issue queries and disseminate top-*k* subscriptions without flooding, CO needs to aggregate the local skylines of associated data tuples of data objects from every node. Every node finds its local skyline and sends it to CO, and then CO stores and makes the index table to use in the next steps.

3.2 A new continuous top-k query

A new query injected to the system at CO must be assigned the initial threshold as the *k*-ranked score of existing data objects in DP and data tuples in the skyline index table. This threshold may or may not be the actual threshold – the threshold of *k*-th ranked items of the entire data objects of every node. However, this is beneficial by testing whether this query with its initial threshold included in DQ and answerable by views but not really injected in the QL and DQ.

1. Query q is not in DQ – This means this query is fully contained in at least one query in DQ. Therefore, the data objects in DP are sufficiently enough to answer, and the initial threshold is equal to the global threshold.
2. Query q is in DQ and answerable by existing views – The data objects in DP are sufficiently enough to answer. However, we decide to construct and disseminate the subscriptions this class of queries.
3. Query q is in DQ but unanswerable by existing views – This means DP is insufficient to answer the query, and the initial threshold is not the global threshold. Therefore, the procedure to request additional data answers is necessary.

In the third case, it frequently occurs on the very first queries in the system. Initially CO does not hold any data objects except the skyline index table. To request only necessary data objects, CO tries to estimate the global threshold by issuing small-sized 1-dimensional histograms in response to a specific query q and aggregate them as the summary of global data distribution. The constructed histograms contain $\lceil \sqrt{q.k} \rceil$ bins which are equi-width ranging between $score_{min}$ (the initial threshold from DP as well as tuples in skyline index table) and $score_{max}$ (this can be correctly calculated by the tuples in skyline index table). We sum the cumulative frequency (CF) from higher-value bins. We stop at the bin that gives $CF \geq q.k$, and the estimated threshold is set at that bin lower limit. We redo this again until it reaches one of stopping conditions as follows; (I) $CF < (1+\gamma)q.k$, (II) The estimated threshold of 2 iterations remains unchanged, and (III) It exceeds the maximum limit. This procedure may take multiple iterations to be completed. After this procedure, the estimated threshold is acquired.

However, for the latter incoming queries, the threshold estimation is possibly inessential if the answers of previously-posed queries in DP can sufficiently answer it (case II and III).

3.3 Top-k subscription dissemination and answer acquisition

CO constructs the top-*k* subscriptions of only queries in DQ. These subscriptions are disseminated to local nodes. The nodes need to send to CO the data objects whose scores according to

the received subscriptions are above the threshold. Because we have discussed that only a small number of nodes taking part in contributing final answers, to prevent flooding to all nodes, we selectively issue the top-*k* subscriptions to nodes whose skyline is crossed with the constructed top-*k* subscription.

$$\text{cross}_{SK_i}(\text{S.sf}, \text{S.ths}) = \begin{cases} \text{true}, & \exists \text{d} \in \text{SK}_\text{i}(S.sf(d) \geq S.ths) \\ \text{false}, & \text{otherwise} \end{cases}$$

Any data objects which have been already sent to CO are flagged to avoid redundant data transferring.

3.4 Maintenance on dynamic updates

A query at CO can be arbitrarily inserted or deleted as well as data objects at local nodes. In this section, we explain how to deal with these updates while ensuring the consistency of the queries' answers.

3.4.1 Query insertion

A new query is handled by the procedure explained in Section. 3.2.

3.4.2 Query deletion

When query q is deleted from CO, we can simply remove q from QL. However, if query q is in DQ, we inform the nodes maintaining the subscription of this query to drop out this subscription. Then DQ in CO needs to be updated, and it is possible that some queries in QL become new dominating queries in DQ. Consequently, new top-*k* subscriptions according to new dominating queries are necessarily disseminated to some needful nodes (Section 3.3)

3.4.3 Data insertion

A new inserted data object at a node may or may not be included in the final answers of the query in QL. To determine the necessity of sending this data object, we have to consider the following cases

1. In the case that a new inserted data object is matched to some active top-*k* subscriptions received so far, we need to send this to CO.
2. In the case that a new inserted data object becomes a new skyline point, this node must send its associated data tuple of this new data object to update the skyline index table at CO.
3. Otherwise, this node keeps the new data object without sending to any node. Since this case is more likely to happen than other cases, a lot of communication cost can be saved.

3.4.4 Data deletion

Obviously, if the deleted data object was already sent to CO so far (indicated by a flag), the node must inform CO to invalidate the data object from DP. Furthermore, the node must update its local skyline if the data object is included in the local skyline. In the case that the deleted data object is on the local skyline, besides informing CO to withdraw it from the skyline index table, the node also needs to send the associated data tuple of new skyline points as well.

Data insertion and data deletion may cause the changes of the actual threshold of queries in QL and DQ. Consequently, the thresholds of the disseminated top-*k* subscriptions so far become inconsistent. For ensuring the correctness of final answers, CO has to selectively re-issue the changed top-*k* subscriptions to the local nodes again with the same procedure in Section.3.3.

4. Performance evaluation

4.1 Experiment setup

We conduct the experiment by implementing the event-based simulator. We assumed that CO can directly communicate with any local nodes. Each node initially holds 150 data objects. The system initially injects 1000 queries. Weightings of the queries are uniformly random as well as parameter *k* which is varied in $[1, \text{k}_{max}]$. Subsequently, we simulate the dynamicity of data and queries as events. The number of events is 200,000 by giving chance of data insertion and deletion at 49% each and query insertion and deletion at 1% each.

We record the final cost of communication defined by the volume of transferred data between CO and nodes. We defined the size of a floating point and an integer equal to 8 and 4 bytes respectively.

4.2 Datasets

1. **Clustered dataset (CL**): Each node randomly draws a coordinate of its centroid of a cluster. Data values on each dimension are independently generated by Gaussian distribution with a specified variance.
2. **Uniform dataset (UN):** Data values in each dimension of each data object are uniformly distributed.
3. **Anti-correlated dataset (AN):** The dataset is generated referring to [3]. Each data record in this dataset will be good at only either dimension. Therefore, the cardinality of skyline and K-skyband must be high.

4.3 Benchmarks

We use the following methods for comparing the performance.

1. **Centralized method (CEN):** The baseline which all data objects and updates are sent to CO regardless of queries
2. **Enhanced Scheme proposed in [2] (ES):** This method applies the filter-based pruning to reduce the unnecessary data updates. This method takes only parameter k_{max} into account.
3. **K-skyband (SKYB):** The data objects which belong to k_{max}-skyband of all local nodes are sent back to CO. This method also concerns only parameter k_{max}.
4. **Our proposed method (THSUB):** Our proposed method described in Section 3 by defining $\gamma = 0.5$.

4.4 Simulation results

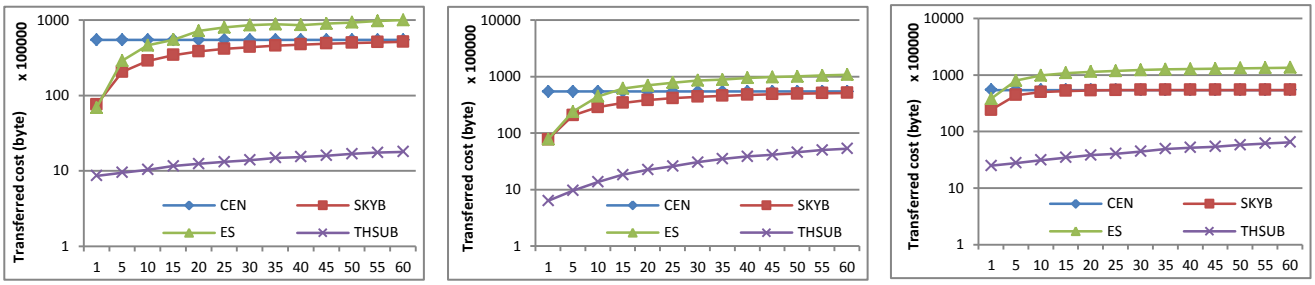

Figure 4: The results of CL dataset (left) UN dataset (middle) and AN dataset (right) on varying k_{max}

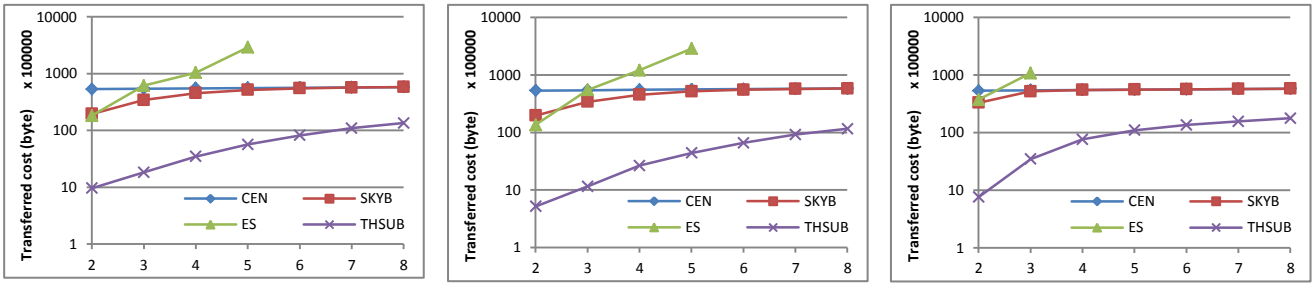

Figure 5: The results of CL dataset (left) UN dataset (middle) and AN dataset (right) on varying dimensionality m

4.4.1 Impact of the number of desired data objects (k)

We evaluate the performance on varying the number of desired data objects or value k defined in a query. The results of CL, UN AN dataset are shown in Fig.4.

According to the result for the CL dataset, for very small k_{max} (roughly 1 to 15), the ES and SKYB methods perform better than the CEN method. Because a lot of unnecessary data objects as well as data updates which are definitely not included in answers can be pruned. However, the cost of the SKYB method growing to be the same as the CEN method and the ES method's cost becomes over the CEN method in higher k. This is because, a part from sending answer candidates, the ES method has to pay the cost for flooding the filter which is expensive. In the contrary, the THSUB method is more efficient because the cost of THSUB method linearly increases with k_{max} and much lower than other three methods. Due to limiting the number of candidates to be sent to CO by the subscriptions, only necessary data objects and updates are sent to CO.

In the UN dataset, the cost of the THSUB method grows slightly faster than the CL dataset; because, for uniform distribution, besides updates are more likely to affect the answers, the subscription update also frequently occurs.

In the AN dataset, the SKYB, ES and THSUB methods suffer since the cost of initialization as well as the size of top-*k* candidates such as skyline points, K-skyband and filter size are object to be very large. The cost of the SKYB and ES methods are almost the same with the CEN method when k_{max} = 15 and k_{max} = 5 respectively. Nevertheless, the THSUB method still outperforms those comparative methods in all cases.

4.4.2 Impact of dimensionality

In this setting, we study the effect of increasing dimensionality. Practically, increasing dimensionality, the size of K-skyband in the SKYB method, filters in the ES method and skyline indexes in the THSUB method increases in the same way because the number of candidates is exponentially increased.

According to the results expressed in Fig.5, in all datasets, the growth rate of the ES method's cost is high. It is noted that it outperforms CEN method only 2-dimensional datasets due to the cost of filter flooding. The benefit of using the SKYB method becomes indifferent from using the baseline CEN method in higher dimensional datasets especially in the AN dataset. We omitted some result of ES methods because it performed worse than the others and it takes too much time to compute. The cost of THSUB method also significantly increases; however, it outperforms those comparative methods by a magnitude.

5. Conclusions

In this paper, we discussed the problem of continuous top-k query processing in distributed environments. We aim to reduce the communication cost due to the top-*k* query processing measured by the volume of transferred data between local nodes which store a fraction of data objects and the coordinator server (CO) where the end-users issue queries. Our method makes use of the initialized skyline indexes and previously-posed queries to cut off unnecessary overhead. Moreover, our method lets local nodes be able to identify the needful updates to inform CO by using top-*k* subscriptions. We conduct the experiment to measure the performance of our method comparing to the comparative methods. The results show that our proposed method significantly outperforms the other comparative methods.

References

[1] Gong, Zhenqiang, et al. "Efficient top-k query algorithms using k-skyband partition." Scalable Information Systems. Springer Berlin Heidelberg, 2009.
[2] Jiang, Hongbo, et al. "Continuous multi-dimensional top-k query processing in sensor networks." INFOCOM, 2011.
[3] Borzsony, S., Donald Kossmann, and Konrad Stocker. "The skyline operator." ICDE, 2001.
[4] Vlachou, Akrivi, Christos Doulkeridis, and Kjetil Nørvåg. "Distributed top-k query processing by exploiting skyline summaries." Dist. and Parallel Databases 30.3-4 (2012): 239-271.
[5] Das, Gautam, et al. "Answering top-k queries using views." VLDB, 2006