Research on Continuous Preference Query Processing in Distributed Environments

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2017

Kamalas UDOMLAMLERT

List of Publications

1. Journal Paper

1. Kamalas Udomlamlert, Takahiro Hara, and Shojiro Nishio: Subscription-based Data Aggregation Techniques for Top-k Monitoring Queries, World Wide Web, (accepted).

2. Kamalas Udomlamlert, Takahiro Hara, and Shojiro Nishio: Threshold-based Distributed Continuous Top-k Query Processing for Minimizing Communication Overhead, IEICE Transactions on Information and Systems, Vol.E99-D, No.2, pages 383–396 (2016).

2. International Conference Paper

1. Kamalas Udomlamlert, Takahiro Hara, and Shojiro Nishio: Communication-Efficient Preference Top-K Monitoring Queries via Subscriptions, in Proc. of the 26th International Conference on Scientific and Statistical Database Management (SSDBM), pages 44:1–-44:4 (2014).

2. Kamalas Udomlamlert, Takahiro Hara, and Shojiro Nishio: Candidate Pruning Technique for Skyline Computation Over Frequent Update Streams, in Proc. of the 26th International Conference on Database and Expert Systems Applications (DEXA), pages 93–108 (2015).

3. Kamalas Udomlamlert, Cosmas Krisna Adiputra, and Takahiro Hara: Grand Challenge: Monitoring Top-k on Real-time Dynamic Social-network Graphs, in Proc. of the 10th ACM International Conference on Distributed and Event-Based Systems (DEBS), pages 317–321 (2016).

4. Kamalas Udomlamlert and Takahiro Hara: Reducing Expenses of Top-k Monitoring in Sensor Cloud Services, in Proc. of the 10th ACM International Conference on Distributed and Event-Based Systems (DEBS), pages 187–198 (2016).

3. Domestic Conference Paper (with review)

1. Kamalas Udomlamlert, Takahiro Hara, and Shojiro Nishio: データ更新が頻繁な環境におけるスカイラインデータ計算アルゴリズムの提案, in Proc. of IPSJ DPS Workshop, pages 17–25 (2014).

4. Domestic Conference Paper (without review)

1. Kamalas Udomlamlert, Takahiro Hara, and Shojiro Nishio: Continuous Top-k Query Processing on Horizontally-Distributed Data, in Proc. of Forum on Information Technology, pages 379–382 (2013).

2. Kamalas Udomlamlert, Takahiro Hara, and Shojiro Nishio: Reducing Expenses of Sensor-Cloud Services for Dynamic Skyline Monitoring, in Proc. of DEIM Forum, online (2016).

Abstract

We are witnessing the era that many things around us (for example, smartphones, watches, cars and cameras) start generating data likewise sensors in traditional sensor network, and especially these are mostly generated in a distributed manner. Many of instruments continuously create data in real-time and become vast streams of data. Monitoring on these streams of data enables many promising and useful applications such as environmental monitoring, smart cities and industrial monitoring. However, it is no use exploring everything in a huge flood of data. On the contrary, end-users are more interested in and want to get informed only a few of most important data that match to their preference (objectives). Therefore, continuous preference query processing does the job to return only a small number of potential data from immense data space. Although, the straightforward implementations of continuous query processing can be achieved by repeatedly executing snapshot query processing of such queries which have been extensively studied, most of them do not perfectly fit because they cannot reduce the critical costs that matter to distributed environments.

In this thesis, we focus on monitoring of two fundamental preference queries including top-k and skyline queries. We begin with the proposal of subscription-based top-k query processing in 2-tier distributed systems, as often seen in traditional sensor networks and peer-to-peer systems (P2Ps). Even though there are existing solutions for this problem, they fail to efficiently reduce the number of sent data or yield too much overhead to support users in a large scale. Our proposed method not only keeps the number of sent data low by using subscriptions but also carefully minimizes the number of subscriptions to be maintained. As a result, the total communication cost, which is a severe factor in sensor networks, can be largely reduced. The results through simulation experiments confirm the advantages of our proposed method over the existing methods.

In skyline processing, a single data update may totally change the skyline set (result set); therefore, for monitoring purposes, it needs to be re-computed periodically to detect changes of the skyline set. However, skyline query processing is a computationally intensive task. Therefore, in the second part of the thesis, we

propose an efficient algorithm to process skyline monitoring queries on frequent updates. Minimum bounding rectangles (MBRs) are used to summarize data movement in each snapshot. This approach enables us to identify a smaller set of candidates for skyline computation. Through the experiments, our proposed method shows the better results in terms of total execution time (computation cost) compared with other methods.

Regarding scalability, the way we manage, collect and utilize data recently has been altered to cloud (a specialized form of distributed computing). Everything in the cloud has been abstracted and provided as convenient services on a pay-as-you-go basis, and data access is not an exception. Running continuous preference queries that need to access a lot of data for comparisons in such a system is high-priced (expense or monetary cost). In the last part of the thesis, we describe cost-minimizing methods for monitoring queries in cloud (*i.e.*, sensor cloud) where the cost is denoted by the expense of data access. Instead of requesting all the latest data in each timestamp, we present a cost-minimizing framework for top-k monitoring – a novel ε-top-k query processing delivering approximate top-k answers with a probabilistic guarantee. The extensive experiments on the real-world datasets demonstrate that our approach can reduce the expense by more than half with desirable accuracy. In addition, we show that the underlying concepts of this method can be easily extended to be used in skyline monitoring, which also gives the same promising results as that of top-k monitoring.

Contents

Chapter 1

Introduction

Nowadays, there are increasing demands to store massive yet complex data. The magnitude of data to be stored becomes impossible to fit in a single machine, so modern data storage for dealing such big data is becoming more distributed (distributed data stores), *i.e.*, data records are partitioned and stored in multiple machines (in this thesis, we call them nodes). Lately, not only data storage but also data generation also turns distributed. In other words, the distributed nodes start generating or updating their own data as time passes (streaming data sources). The most obvious system that share this behavior is sensor networks (SNs) where a number of sensors perform data sensing and enable us to acquire various kinds of latest information from physical world in real time.

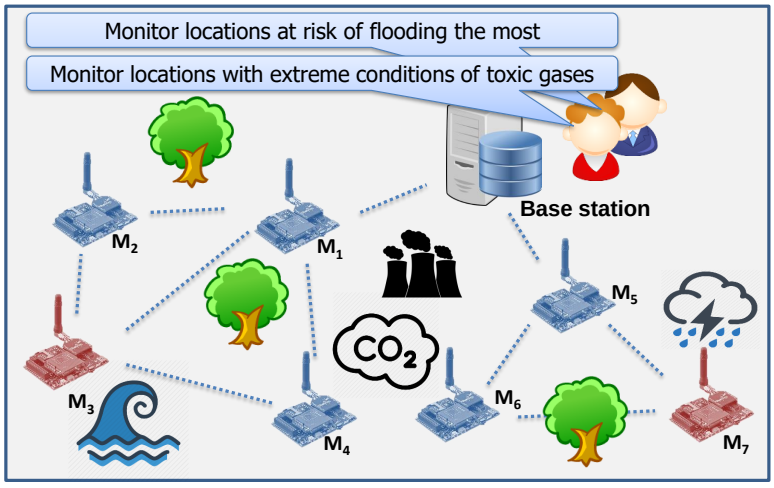


Figure 1.1: Sensor networks for environmental monitoring

The conventional way to perform data processing is to aggregate generated data from sensors (nodes) to a centralized server called the base station (BS), then end-users can derive benefits by performing data analytic as shown in Fig. 1.1. Even though we often exemplify scenarios of sensor networks in this thesis, this architecture can be generalized to highly distributed systems (client/server systems, hybrid peer-to-peer (P2P) systems and other collaborative distributed systems).

Due to the outgrowth of these sensors including public/private sensor networks and smart phones via participatory sensing [18, 23, 43], they create a massive amount of data records continuously. Data sensing from sensors in each location is associated with an identifier (*e.g.*, id and location), timestamp and multiple values (*i.e.*, data records, multidimensional data, data objects or data tuples). For example, a sensing report from a weather station in San Francisco at 10.00AM consisting of temperature from a thermometer sensor and humidity from a hygrometer sensor. In addition, in its nature, these data are generated in a distributed fashion, *i.e.*, distributed data sources.

Even though these nodes take actions on data generation and data acquisition, but not all data are created equal. In other words, we normally do not look with attention into every single data that are generated. Especially, in the era of Big data [4], it would be very difficult, if not impossible, to do so. On the contrary, we are more likely to observe or monitor only data that truly have high impacts and values to draw any useful conclusions. These conclusions can lead us to better decision making. Besides, such monitoring applications need to be done in real time. The examples of crucial applications include as follows:

1. Landslide and avalanche prevention– monitor of soil moisture, snow level and vibrations to evaluate risks in land conditions

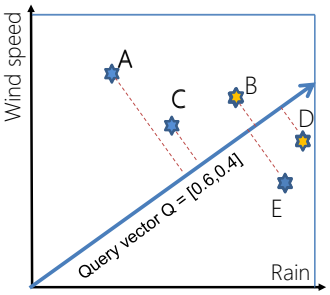
2. River flood risk monitoring– monitor of water level, water flow and precipitation rate to monitor highly-hazardous locations

3. Air pollution monitoring– monitor the high values of dangerous toxic pollutants of air conditions in diverse areas

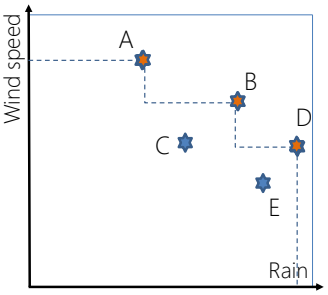
If these monitoring systems are effectively deployed, life-saving countermeasures can be properly imposed to prevent catastrophic situations. To perform these tasks, preference query processing plays important roles on qualifying and returning only a

	Rain (x_1)	Wind (x_2)	$0.6x_1 + 0.4x_2$
A	39.18	83.25	56.808
B	73.08	68.75	71.348
C	54.91	54.25	54.646
D	94.87	53.85	78.462
E	82.97	38.05	65.002

(a) Table of data tuples



(b) Example top-2 query



(c) Example skyline

Figure 1.2: Top-2 includes B and D while skyline set includes A, B and D

small number of potential data from massive generated data to users based on users' preferences or objectives. It helps users narrow down the scope of abundant data and provides the insights about how to deal with the situations that are happening.

To specifically illustrate, for example, given some examples of the air pollution monitoring, experts may need to keep an eye on some distinct conditional locations using queries as follows:

1. Top-3 locations that contain the combinations of 6 common dangerous air pollutants[1] the most

2. All locations that contain dangerous air pollutants in unique extreme conditions (Pareto optimal candidates)

The first exemplified query is classified as a top-k query. By using top-k query processing, users can acquire k-best score data with regards to a pre-defined preference scoring function (*i.e.*, weightings of each attribute), where k and a scoring function are the input parameters from users. This helps users limit the number of returning results (at most k), which are best matched to the given objective. Fig.1.2b shows an example of top-k query processing at one snapshot over multi-attribute data in Table 1.2a.

The second query is called a skyline query where the problem is long known as the Maxima vector problem [26, 27, 41], which is directly related to the concept of Pareto efficiency in economics and business management field [51]. In this query, none of parameters is required except for a set of attributes. One of the useful properties

[1]https://www.epa.gov/criteria-air-pollutants

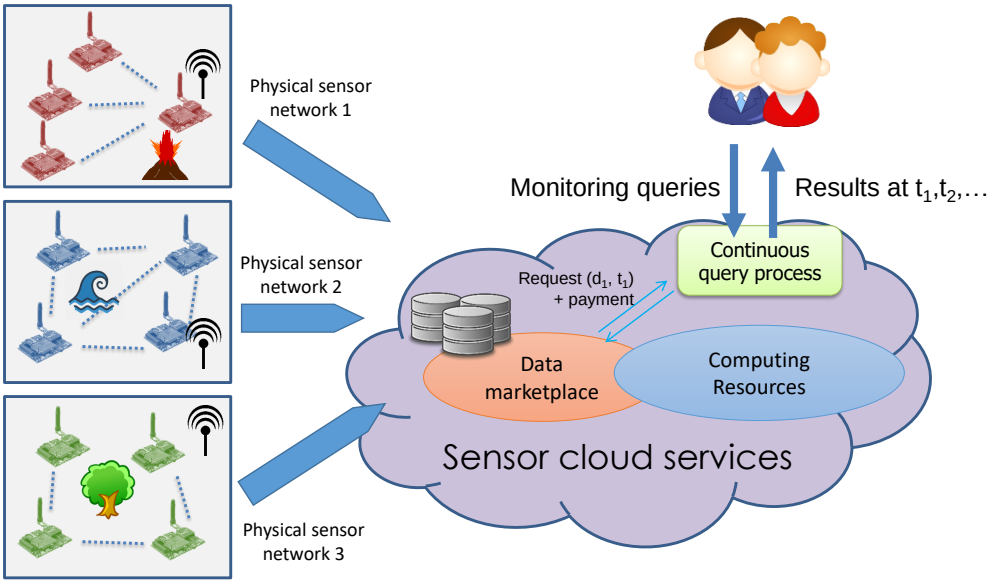


Figure 1.3: Sensor cloud services where users can easily request shared paid sensor data

of this query is that a skyline set (a set of skyline data points) contains any possible top-1 answers (for monotonic scoring functions), so skyline monitoring gives users the insight of all possible extreme conditions dynamically. This will be explained in detail in Chapter 3. It is noted that the size of a skyline set mainly depends on the number of attributes (dimensionality) and data distribution, nevertheless the size of a skyline set is expectedly much smaller than the size of entire dataset. Fig.1.2c shows an example of the skyline set over data points in Table 1.2a.

1.1 Research Issues

A large number of works have focused on diverse variants of query processing methods in centralized databases and distributed databases, but these query processing methods for monitoring purposes introduced in the previous section, where their executions continuously run on dynamic data in distributed environments, still have many open issues to be studied. Even though we can employ some of existing works for snapshot queries (as opposed to continuous queries) in the context of continuous queries by repeating it every short period of time, but they are inappropriate and raise problems including communication cost and high response time. In addition, because data management

changes gradually with modern technologies, distributed environments here cover not only the traditional schemes of highly distributed environments (*e.g.*, centralized P2P systems and sensor network deployments illustrated in Fig.1.1) but also cloud-extended infrastructure, *e.g.*, Sensor cloud [3, 48] where sensor data are aggregated from diverse sensor networks, shared and accessed in the cloud (Fig.1.3). The latter setting, to the best of our knowledge, this thesis is the first one to addressed this issue.

These settings bring a couple of major challenges because of some limitations that are worth considering as follows:

1. Some distributed systems especially wireless sensor networks (WSNs) have limited data bandwidth and limited battery capacities, which are majorly consumed by their data transmission components. Large communication overhead among nodes from performing many continuous queries, which is referred to data transfer cost, can negatively affect scalability, network lifetime and response time.

2. The performance of the base station (or servers) may slow down resulting in higher response time if it needs to compute a large amount of aggregated data for answering continuous queries (computation cost).

3. Cloud-based infrastructure, *e.g.*, sensor cloud services, is based on pay-as-you-go basis; running continuous queries can be costly in terms of the number of data accesses (data access cost).

Therefore, in this section, three research objectives are described to alleviate those limitations for preference query processing in such distributed environments.

1.1.1 Data Transfer Cost Reduction

A communication component of a sensor node, which takes a role of receiving and sending data messages among nodes and the base station, consumes non-trivial amount of battery power [19, 33, 67]. In order to carry data to the base station, intermediate nodes (*e.g.*, nodes M_1 and M_5 in Fig.1.1) take more burden relaying data on behalf of other nodes to the base station, and are likely to run out of battery first. Running out of battery of such nodes in the system negatively affects network connectivity as well as overall network lifetime. In P2P systems, this can cause data bottlenecks at such nodes

while excessive data transfer is a major cause of packet losses in systems, for instance, MANET [40]. Furthermore, to maximize energy efficiency is one of the best practices for green computing [89].

Therefore, there is a demand for efficient continuous query processing methods in distributed environments that are able to save the cost of data transfer by avoiding sending redundant data objects, messages and overhead involved in query processing (for example, data filters [91] and subscription messages in publish-subscribe-based query processing [58]).

1.1.2 Computation Cost Reduction

In the perspective of the base station, even though the base station is often assumed to be more powerful than other nodes in both terms of computing and power resources, the base station must be able to rapidly process frequent streams of data coming from many nodes and finally report up-to-date continuous query answers to the end-users. Failing to do so brings about high response time and inability to detect or report the critical events to users in time, although the specifications of latencies of data updates in streams and response time are varied based on the types of applications. This problem can be alleviated by designing efficient algorithms to handle frequent data updates at the base station. It is noted that pruning out the irrelevant data to be sent to the base station at the edge (shared the same idea with Section 1.1.1) is also helpful in computation cost reduction.

1.1.3 Data Access Cost Reduction

Data access cost must be taken into account when executing continuous preference queries in highly abstract distributed environments, for example, sensor cloud where all physical development details have been abstracted and made easy for users to request desirable data objects on demand as a service. The provided service accepts requests from users and responds with values or data tuples with regards to the specified requests.

Most of preference queries are rank-aware queries, *i.e.*, result determination involving positions (ranking) or interactions among data in the dataset. To be aware of the positions or ranks of all data is to pay the price for requesting all data and then per-

form sorting or comparisons, even though, after all, some paid data are not included in the final results. In commercial sensor cloud, this incurs a huge sum of expense (*i.e.*, monetary cost, payment to data providers) especially when running continuous queries, which keep requesting data for every timestamp.

1.2 Research Contents

Subscription-based continuous top-k query processing

In Chapter 2, we address the issues when using the concept of publish-subscribe (pub/sub) model to aggregate only data that are potentially matched to the end-users' preferences for processing continuous top-k queries in distributed environments such as sensor networks and peer-to-peer systems. In the scheme, nodes are taken as publishers, and users at the base station (centralized server) are taken as subscribers. We propose our subscription-based continuous top-k query processing requires 2-round communication (2 rounds of subscription dissemination) between the base station and nodes. Then, we show that the number of subscriptions can be further reduced to save data transfer (data transfer cost or communication cost). We introduce the concepts of a set of dominating subscriptions and a set of minimal subscriptions. Instead of disseminating all subscriptions of all queries, disseminating only a set of minimal subscriptions, which is much smaller, is enough to guarantee the completeness of the answers. Additionally, adaptive strategies to renew subscription are proposed to avoid excess maintenance cost.

Skyline monitoring for frequent update streams

Skyline computation is a computationally intensive task – high computation cost, and a single data update can totally change the final skyline set. In skyline monitoring where many cases require the response latency to be small as close as real-time, it is obviously not effective to naively calculate skyline result from the scratch upon every data update. In Chapter 3, an efficient algorithm to fast process up-to-date skyline on incoming frequent data steams at the base station is proposed. The main concept is to use minimum bounding rectangles (MBRs) to dynamically prune out unnecessary candidates from skyline re-calculation. In addition, the proposed strategy for maintaining MBRs

enables our approach to be adaptive to the changes of data dynamicity resulting in faster computation.

Cost-minimizing methods for preference query monitoring

As stated in Section 1.1, due to the advancement of internet connectivity, the paradigm of how we manage sensor data has shifted to the cloud. Even though its hidden underlying implementations are still distributed, query processing from the viewpoint of users on such systems has data access cost which is different from the traditional sensor networks.

In Chapter 4, we propose a variant of a top-k query called ε-top-k query which returns an approximate result set with controllable errors for continuous queries while the cost of data access can be significantly reduced.

Clearly, more complicated calculations bring about more computing complexity. We also enhance our methods by using cache pruning techniques and show that this expensive computation can be alleviated by well-known distributed and parallel computing framework, *i.e.*, MapReduce and Spark. In addition, using the same fundamental of ε-top-k query, we can easily extend this idea for continuously computing approximate skyline in lower cost called ε-skyline.

1.3 Organization of Thesis

This thesis consists of five chapters, and the rest of this thesis is organized as follows.

In the next chapter, we introduce and discuss about continuous top-k query processing in distributed systems in Section 2.1 and review the prior works related to this topic in Section 2.2. Section 2.3 describes the preliminaries of our work including our assumed system architecture, query model and problem definitions. Our proposed methods for continuous top-k query processing, which are based on publish-subscribe models, and the techniques to fast identify the minimal set of subscriptions to decrease the number of subscriptions resulting in significant lower communication overhead are described in Section 2.4 and Section 2.5 respectively. The experiment setup and the simulation results to exhibit the performance of our proposed methods are described and discussed in Section 2.6. Finally, we conclude our work in this Chapter in Section

2.7. The research in this chapter is based on our works published in [79, 80, 81, 85][2].

In Chapter 3, we explain the skyline monitoring problem and its issue for processing frequent update streams in Section 3.1 and then review the related work in this topic in Section 3.2. Section 3.3 explains the preliminaries of our work including data model, the properties of minimum bounding rectangles (MBRs) and their usage. Then, Section 3.4 describes our proposed algorithm for skyline monitoring aiming at speeding up computation. Our proposed method has advantages over other methods in terms of adaptivity over dynamic data. We compare our proposed method by conducting the experiments using both synthetic and real datasets. The details are discussed in Section 3.5. Finally, this chapter is summarized in Section 3.6. The research in this chapter is based on our works published in [82, 83][3].

In Chapter 4, unlike Chapter 2 and Chapter 3, we consider continuous query processing in more-abstracted distributed architectures, *e.g.*, sensor cloud services. We firstly introduce the motivation and point out some given example scenarios in Section 4.1 as well as the related work in Section 4.2. The system architecture, its data model, its data access model and the query model are exemplified in Section 4.3. In Section 4.4, we illustrate our ideas as well as the basic framework on how to achieve lower data access cost. Then, we propose two algorithms and also their enhancements in Section 4.5 and Section 4.6 respectively. The experiments are conducted to examine the benefits of our proposed algorithms in Section 4.8. In addition, we also briefly introduce the approach to extend our proposed methods for skyline monitoring, and we also show the obtained results for that application in Section 4.9. Finally, we conclude Chapter 4 in the last Section 4.10. The research in this chapter is based on our works published in [78, 84].

[2][85] is available at Springer via http://dx.doi.org/10.1007/s11280-016-0385-1

[3][83] is available at Springer via http://dx.doi.org/10.1007/978-3-319-22852-5_9

Chapter 2

Subscription-based Continuous Top-k Query Processing

2.1 Introduction

In recent years, the amount of digital data is surging tremendously because of the rapid population of mobile and sensor devices, which continuously generate and store those generated data. Therefore, to process a top-k query which is a rank-aware query over those distributed data sources, naively we need to aggregate all those distributed data to a single server. In the perspectives of green IT or some deployments, *e.g.*, wireless sensor networks, the constraint of bandwidth consumption, which directly affects battery lifetime, is often stressed. Therefore, it is not feasible to aggregate all data. The paradigm is shifted to the concept of query-then-store which users define their set of top-k preferences beforehand and then the system will aggregate only a sufficient set of data from multiple data sources. Due to a characteristic of a top-k query, a rank-aware query, it is challenging to request a partial set of data from the entire data without losing the accuracy of the delivered final answers. In addition, more complicated requirements of data model and query model are also taken into account.

In this chapter, we address the problem of continuous top-k data aggregation given by an example application in Fig.2.1. A possible application of this research is to monitor the latest weather information to record only some attentive events in real-time, *e.g.*, the risk of heavy rain, heatstroke, storm surge and avalanche. The weather infor-

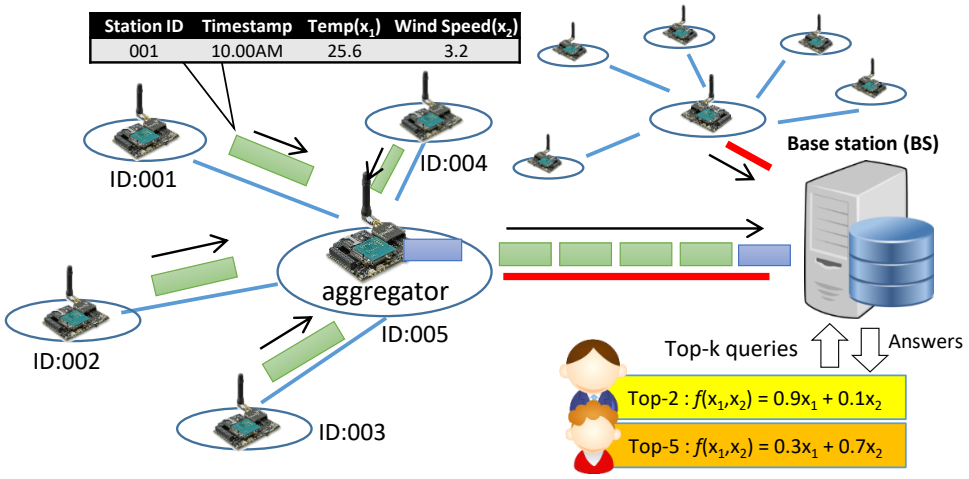


Figure 2.1: Example scenario: data aggregation via aggregators to answer top-k queries registered at the base station (BS)

mation of each site is composed of multiple numerical values (multi-dimensional data) including temperature, wind speed, precipitation, snow depth, humidity and so forth in each timestamp, and the attentive events are evaluated from this information by giving different weightings on each factor. Traditionally, these sources of the weather data, *e.g.*, weather stations, need to report their own real-time data values to the centralized server, and these values keep changing every fixed time period, for example, record every 10 minutes. The officials at the centralized server keep observing and prioritizing the first few sites which yield high potential of occurrences with the additional analysis from various factors by the experts, while the rest of the list seems benign. Top-k query processing is helpful in this task to report only k most attentive sites in response to the query preferences. Furthermore, this kind of situation can be seen in a modern system like smart meters which are setup at each household to measure and report the electric, water and gas usage to be analyzed and for improving the efficiency and reliability of resource and supply distribution.

While the experts tend to be interested in a few candidates at a time, the centralized scheme which aggregates all data readings back to the centralized server in every epoch is impractical in both terms of communication cost and scalability. Traditional snapshot top-k query processing is definitely inefficient in this case because, by dealing a lot of queries, it can incur a large delay as well as redundant data records to be returned.

In a large scale deployment such as P2Ps, data are sent by relaying from data sources to a centralized server (a base station). Therefore, in each epoch, an intermediate node which helps relay data to the base station will incur more load than others. Such intermediate nodes can be found as cluster heads in a cluster topology, high-level nodes in a hierarchical topology, nodes which are close to the base station in a mesh topology and even regional data aggregation server on the Internet. Here, we will call a node behaving this as an aggregator. If these aggregators also have same battery power and bandwidth as other local nodes, they are likely to be put off first. This obviously reduces the network lifetime in the system.

To solve the problem above, we propose a method to reduce the number of reported data records which are accumulated at the aggregators before relaying to the terminal base station. Obviously, of all aggregated data, they contain both data records that finally belong to final top-k answers and not ones (larger set). Our method prevents intermediate nodes from relaying unnecessary data to suppress communication cost.

Our method adopts the concept of a publish-subscribe model for continuous queries which refer aggregators as publishers and many users at the base station as subscribers. Therefore, the base station has to issue queries' meta-data to register their areas of interests called subscriptions to each aggregator. Then the aggregators need to send their holding data records which match with one of the registered subscriptions in each timestamp. However, directly adopting this idea is inefficient because the overhead of subscription dissemination can be extremely high when dealing with a large number of queries. Apart from the issue of scalability, subscriptions can become less accurate or invalid if global data distribution has been changed. New subscription reconstruction and new dissemination must be executed, and the additional communication cost due to its maintenance is non-trivial. Perhaps, the cost of subscription maintenance may overcome the cost of transferring data records, so using this scheme naively may not be feasible in the real practice.

Therefore, our proposed method adopts techniques to manage these subscriptions by reducing the number of subscriptions to be disseminated while guaranteeing the completeness of the final top-k answers of all queries. We identify a minimal set of subscriptions instead of sending entire of queries' subscriptions as well as the way to decide an appropriate time to renew those subscriptions to save further communication cost.

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

- We formulate the requirements of continuous top-k data aggregation in distributed data sources.
- We extend the basic concept of publish-subscribe for executing continuous top-k data aggregation with our proposed techniques to save a large amount of communication overhead.
- We conduct experiments in various settings by using both synthetic and real dataset to show that our proposed method can achieve lower communication cost and outperform the other aggregation techniques.

The remainder of this chapter is organized as follows. In Section 2.2, we introduce related work of this chapter. Section 2.3 presents preliminaries explaining system environments, query model and problems. In Section 2.4, we present our proposed method and its enhancement in Section 2.5. In Section 2.6, we discuss the results of the simulation experiments, and in Section 2.7, we summarize the chapter.

2.2 Related Work

Top-k query processing has been widely researched in the database community. Fundamentally, it has been implemented in centralized databases to return only k most favorable ranked records. Many studies aim to improve the efficiency of various kinds of top-k queries by limiting scans, using indexes [42, 96, 101] and cache [6, 11, 20, 30]. Moreover, efficient top-k query processing for data streams has also been researched [50, 65, 94]. All of them are focusing on reducing the number of scans and computation time for fast computation.

In the viewpoints of distributed systems, a top-k query is also demanded especially in peer-to-peer networks (P2Ps) and wireless sensor networks (WSNs). However, different assumptions have been made based on data models, and their requirements. [64, 91] are interested in querying on single-value data while reducing the amount of transferred data. For multidimensional data, they can be mainly divided into two categories including vertically-partitioned data [5, 62] and horizontally-partitioned data. In this research,

we focus on the latter case. In either case, most of studies so far focused on snapshot queries on static distributed data and aim to reduce the amount of transferred data records by using various techniques, for example, caching [86, 87, 100], progressive query [61] and estimation from historical data [67]. In the case of dynamic data which are frequently found in WSNs, [39] proposes a method to prune irrelevant data records by constructed filters and aggregate only a partial set of data.

2.3 Preliminaries

2.3.1 System environments

System model

The distributed network consists of N_A aggregators ($M_1, M_2, \ldots, M_{N_A}$) and a single base station (BS) which is more powerful than aggregators in both terms of computing capability and storage. An aggregator M_i takes a role of accumulating data records from neighbor nodes in every epoch. Therefore, we simply assume that, at each epoch, an aggregator holds a set of data records D_{M_i}. Each aggregator logically connects and collaborates with BS. BS takes a role of taking queries from all end-users and performs data aggregation. BS can be seen as a centralized coordinator server to relay and deliver the final result to end-users. The system architecture is similar to [46], but they aim to monitor skyline queries while we focus on top-k queries.

Data model

Each data record from a local node is attached with id which we can assign in the initialization of the system deployment. Each data record generated at timestamp t is composed of a tuple of $m + 2$ numerical values including id, timestamp and m-dimensional attribute values. Those m-dimensional attribute values are represented in m-dimensional Euclidean space ($(x_1, x_2, \ldots, x_m) \in \mathbb{R}^m$). At timestamp t, each aggregator holds only a partial set $D_{M_i,t}$ of all dynamic data records ($D_t = \bigcup_{i=1}^{N_A} D_{M_i,t}$). It holds that $D_{M_i,t} \subseteq D_t$ and $\bigcap_{i=1}^{N_A} D_{M_i,t} = \emptyset$. In the system, the number of dynamic data records of each aggregator is constant, and only their values are changed. Let N_D stand for the total number of data records in the system, $N_D = |D_t| = |\bigcup_{i=1}^{N_A} D_{M_i,t}|$.

Basically, only m-dimensional attributes of each data record are used for ranking, and we represent it as $d_{i,t} = (d_{i,t}[1], d_{i,t}[2], \ldots, d_{i,t}[m])$ where $i = \{1, 2, \ldots, N_D\}$ and t is a positive number of timestamp. Therefore, $\forall d_{i,t} \in D_t$ and $D_{M_i,t} \subseteq D_t$. It is noted that, at different times t, the values of attributes are possibly changed.

2.3.2 Top-k query model

A multidimensional top-k query $q = (f_q, k_q)$ from a user is defined by a linear scoring function in which its weightings can be represented as a vector $f = (w_1, w_2, \ldots, w_m)$ where w_j stands for positive weighting at j-th dimension while a parameter k stands for the number of desired data records. We focus on linear functions such that a score of a data record d_i is $f(d_i) = \sum_{j=1}^{j=m} w_j d_i[j]$. This class of functions is common because it represents how a user gives priority (weighting) to each factor. A set of weightings can be acquired by many possible ways, for example, given directly by users, adapted from the studies of each objective or suggested by the system via a questionnaire. A scoring function is monotonic, that is if $d_a[i] \leq d_b[i]$ for all $1 \leq i \leq m$, then $f(d_a) \leq f(d_b)$. The value k defines the number of desired data records. Users must define their query preferences a priori to BS, and BS records these queries in the query list (Q).

The final top-k answers of query q at timestamp t are denoted by $T_{q,t} = \{a_{q,1}, a_{q,2}, \ldots, a_{q,k_q}\}$ where $a_{q,i} \geq a_{q,j}$ when $i > j$ and $T_{q,t} \subseteq D_t$. By the definition of final top-k answers, $\forall d_{x,t}, d_{y,t} : d_{x,t} \in T_{q,t}, d_{y,t} \in D_t \backslash T_{q,t} \rightarrow f_q(d_{x,t}) \geq f_q(d_{y,t})$.

2.3.3 Problem definition

To answer all top-k queries stored in the query list Q at timestamp t ($T_{q,t}$ for each $q \in Q$), BS must receive a sufficient set of data records from each aggregator M_i – say L_{M_i} such that $\forall q \in Q : T_{q,t} \subseteq \bigcup_{i=1}^{N_A} L_{M_i}$.

In this research, we aim to minimize the communication cost for answering all top-k queries in Q for each timestamp t. The communication cost for querying involves not only the cost of transferring data items from aggregators ($\sum_{i=1}^{N_A} |L_{M_i}|$) but also the cost of transferring query messages, filters or other communication overhead involving in any specific processing methods, *e.g.*, subscriptions and their updates.

2.3.4 Distributed continuous top-k query processing

Traditional snapshot top-k query processing

In this section, we discuss about traditional distributed top-k query processing methods on horizontally-partitioned databases. Since a top-k query is a rank-aware query, it is very challenging to reduce transferred data in distributed systems. The basic method for a snapshot top-k query is to issue the query from BS to every node in the network. To guarantee the completeness of the final answers, each node must send its own data records of k highest scores back to BS. This procedure can be finished in a single round, but the number of returned data records is kN_A objects. In a large scale system, this is unacceptable because $k(N_A - 1)$ data records not included in the final answers are wastefully transmitted to BS.

The drawback of the method explained above is due to the number of contacted local nodes because it has to contact to all nodes in order to request final answers whether nodes do not contribute final answers.

[86] proposed a method to prevent BS from contacting all nodes by utilizing the concept of skyline points. BS initially caches all skyline points of local nodes' data records, and they are used for deciding the order of nodes to be requested. The query will be processed at one local node at a time by choosing the most promising node first based on the maximum score of its skyline points stored at BS. In the first round, first k items will be returned, and then it compares k-th highest score of data records retrieved so far with the skyline points of the next promising node. If those skyline points of the next are not better than ones retrieved so far, the final results are complete and the processing can be stopped. The trade-off of this method is that it requires multiple iterations to finish (up to k iterations in the worst case), so the latency until the end of processing can be high.

The idea to derive a benefit of caches (previously-posed queries) is found in [61, 100]. [61] proposed a method to identify whether the answers of previously-posed queries processed before are included in final results of a new query. If they are not enough, the query continues requesting remaining data records from local nodes. [100] lets local nodes cache some queries with the answers to quickly response to the popular query patterns. As a result, the number of contacted nodes can be reduced, because the final answers can potentially be reached from neighbor nodes.

All traditional methods described above are not applicable to continuous queries, because they have to repeatedly query (proactive methods) for the latest final results every time epoch, and the appropriate query interval is difficult to be determined. This causes a lot of query messages to be forwarded and redundant results to be returned.

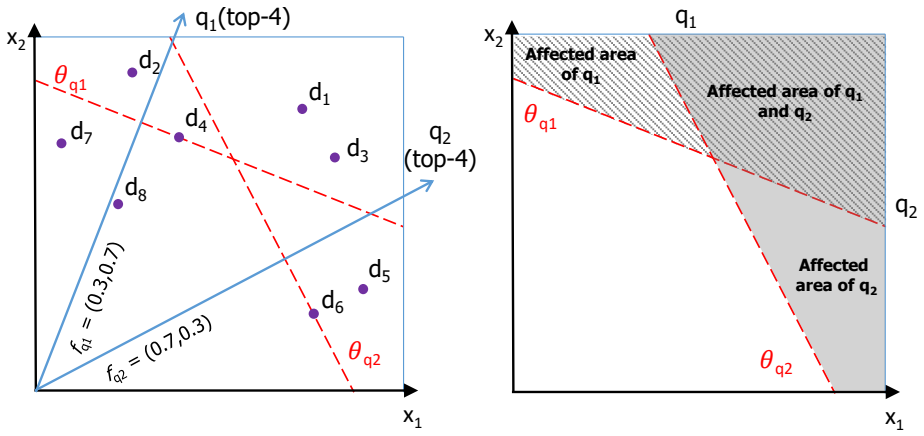
Existing methods for continuous queries

Instead of repeatedly querying like traditional methods for continuous queries, the reactive scheme, local nodes sending only data records which are likely to be included in the final answers back to BS (potential data records), is more promising. In the idea of [87], we can retrieve a part of local dataset from each node which satisfies the completeness of final answers back to BS called K-skyband. K-skyband returns a set of data records that are dominated by at most $K-1$ other data records. An data record dominates another if it is as good as or better in all attributes and better in at least one attribute. The accumulation of these data can sufficiently answer all monotonic top-k queries where $k \leq K$. The definition of K-skyband is referred to [52]. When a data update occurs at a local node, if it affects local K-skyband, that data update must be reported to BS. It can prune a lot of unnecessary updates, but the efficiency becomes worse in a skew-data distribution and a large-scale system.

According to too large K-skyband, the method in [39] solves this problem by using a more efficient global filter. BS retrieves k_{max}-dominant data records first, then it constructs the filter. Any data records which are dominated by this filter is impossible to be included in the final answers of any top-k queries where $k \leq k_{max}$. The same filter will be setup at all local nodes to prevent unnecessary data updates. The obvious drawback is the maintenance cost because, when a data record that is a part of the filter or inside the filter is changed or removed, it gradually invalidates filter points and the pruning efficiency is degraded. As a result, false positive data records are leaked to BS when being used in long terms. The way to handle this problem is to re-construct and disseminate a new global filter to all nodes, but this is costly.

Subscription model

The novel concept, query-then-store, is an alternative method for executing continuous queries. The idea to perform information filtering at information generating sources to



(a) Weighting vectors and their thresholds (b) Affected areas

Figure 2.2: Example of two top-4 queries including q_1 and q_2

prevent overwhelming data updates has been studied in [58, 97] by announcing the specification of interests to the information sources. That set of specifications are defined in subscriptions. However, the content in a subscription depends on query types and data models. Here, a subscription for top-k query processing is bound to each aggregator to inform it of which data updates should be notified to BS since we know that most of generating new data updates are not included in final top-k answers.

Given a set of final results of a top-k query q, top-k answers and non-top-k answers can be easily divided by setting the score of k-ranked data record $(f_q(a_{q,k_q}); a_{q,k_q} \in T_{q,t})$ as the actual threshold $\theta_{q,t}$ as the following equation.

$$\theta_{q,t} = (f_q.w_1)a_{q,k_q}[1] + (f_q.w_2)a_{q,k_q}[2] + \ldots + (f_q.w_m)a_{q,k_q}[m] \tag{2.1}$$

We refer $(f_q.w_1)x_1 + (f_q.w_2)x_2 + \ldots + (f_q.w_m)x_m = \theta_{q,t}$ as the threshold line of query q of timestamp t.

In addition, we define the area where is above the hyperplane $(f_q.w_1)x_1 + (f_q.w_2)x_2 + \ldots + (f_q.w_m)x_m \geq \theta_q$ which is a half space as *the affected area*, because any changes of data values outside this area definitely do not affect the query's final answers as well as the threshold and vice versa. We omit t in $\theta_{q,t}$ or $T_{q,t}$ if the statement does not imply any specific timestamp. An example of two top-4 queries is shown in Fig.2.2.

Seeing that the final top-k answers can be correctly retrieved by using any traditional top-k query processing methods, k-th ranked data record is certainly known. The system

can create a top-k subscription denoted by a scoring function/preference function f_q and a threshold $\theta_{q,t}$ in response to every single query and bind it to every node to suppress the number of transferred data records to be sent to BS. The aggregators only need to send data items $d_{i,t}$ which $f_q(d_{i,t}) \geq \theta_{q,t}$. We call that a data record *matches* with a subscription. In the same way, we also call a subscription matches a data record. Therefore, the number of disseminated subscription messages will be $N_A|Q|$.

None of the researches above concern about query management at BS to improve the performance in terms of communication cost when dealing with many queries for numerous users. Looking into the relations among those queries, some queries (sub-queries) are fully contained in other queries (dominating queries). In other words, one query may be subsumed by other queries (Query subsumption [37, 76]). Being able to identify the relation of these queries can lead to the capability of disseminating only some potential subscriptions instead of subscriptions of all queries. This can save cost of communication and cost of maintenance.

Answering top-k queries using multiple views

Basically BS is able to identify whether a given subjective query can be answered by previously-posed queries' answers (materialized views). For example, let us assume that a list of n_v existing queries $Q_V = \{q_1, q_2, \ldots, q_{n_v}\}$ with their complete final answers is given, each query $q_i \in Q_V$ consists of $(f_i, k_i, \theta_i, T_{q_i})$, and we want to test whether or not subjective query q_* with scoring function $f_* = (w_1, w_2, \ldots, w_m)$ and value k_* can share the answers with the queries in Q_V. The linear optimization method can be used for such that test by finding $x_1, x_2, \ldots, x_m$ that maximizes θ_m using the following program.

$$\begin{aligned}
&\text{maximize } \theta_m = (f_*.w_1)x_1 + (f_*.w_2)x_2 + \ldots + (f_*.w_m)x_m \\
&\text{subject to } (f_1.w_1)x_1 + (f_1.w_2)x_2 + \ldots + (f_1.w_m)x_m \leq \theta_1 \\
&\qquad\qquad (f_2.w_1)x_1 + (f_2.w_2)x_2 + \ldots + (f_2.w_m)x_m \leq \theta_2 \\
&\qquad\qquad \vdots \\
&\qquad\qquad (f_{n_v}.w_1)x_1 + (f_{n_v}.w_2)x_2 + \ldots + (f_{n_v}.w_m)x_m \leq \theta_{n_v} \\
&\qquad\qquad x_1, x_2, \ldots, x_m \geq 0
\end{aligned} \tag{2.2}$$

If $\theta_m \leq \theta_*$ where θ_* is the k-ranked score of the available data records from the

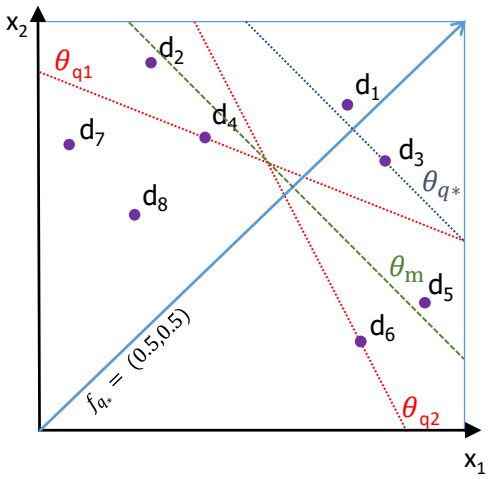


Figure 2.3: Example of 2 materialized views (q_1, q_2) and a subjective query (q_*)

queries in Q_V ($\bigcup_{i=1}^{n_v} T_{q_i}$), then the queries in Q_V subsume q_*. In other words, q_* is answerable by the answer sets of existing queries Q_V, *i.e.*, $T_{q_*} \subset \bigcup_{i=1}^{n_v} T_{q_i}$ without additional data retrieval from distributed data nodes.

To illustrate, given 2 existing queries from Fig.2.2a and a subjective query q_* with $f_{q_*} = (0.5, 0.5)$ and $k_{q_*} = 2$, the resulting θ_m w.r.t q_* is as shown in Fig.2.3 while $\theta_{q_*} = f_{q_*}(a_{q_*,2}) = f_{q_*}(d_3) \geq \theta_m$. Hence, $\{q_1, q_2\}$ subsumes q_* and $T_{q_*} \subset \bigcup_{i=1}^{2} T_{q_i}$.

The methods to improve the executing performance by proposing view selection algorithms and indexing techniques have been studied in [20, 92].

2.4 Proposed Algorithms

2.4.1 Main idea

According to the assumed environment and its application, data records are aggregated periodically at every periodic timestamp to answer the posed continuous queries. Traditionally, BS aggregates entire data records. However, receiving all data records as mentioned above is too expensive. To prevent BS from receiving unnecessary data records, BS disseminates a set of subscriptions which contains scoring functions and thresholds of each query. This is to limit the number of returned data records for the next iteration.

The concept of publish-subscribe scheme for continuous queries introduced in Section 2.3.4 can be implemented in BS and aggregators by initially aggregating all data records at timestamp $t = 0$, and BS constructs a set of subscriptions of the queries in Q and disseminate them to aggregators.

After that, on a new periodic update at timestamp t where $t > 0$, aggregators will send a set of data records which are potentially included in top-k answers to BS based on a set of existing subscriptions. Once BS receives this set of data records, BS checks whether the set of subscription needs to be updated. If yes, this also means the set of data records is possibly insufficient to answer top-k queries. Then, a set of updated subscriptions is disseminated to aggregators (first round of subscription dissemination).

Once aggregators receive a new set of subscriptions, they send an additional set of data records to BS based on the new set of subscriptions. The same procedure at BS is repeated when it receives the additional set of data records. If the data records BS has received so far are still insufficient, another time of subscription from BS may be required.

The algorithm for aggregators and BS after $t = 0$ are described in Algorithms 1 and 2 respectively. We later proof that, for each iteration t, BS requires at most 2 times of subscription dissemination (2 rounds of communication) to be able to sufficiently answer top-k queries.

2.4.2 Algorithm explanation

On a new periodic update of timestamp t, aggregators need to execute the procedure in lines 4-10, Algorithm 1 to report latest updates of data records to BS. We select data recorded to be included in the set of data records to be sent to BS (L_{M_i}) with 2 following criteria. *a*) The data records which matched with at least one subscription in the subscription list of the previous iteration ($t - 1$); and *b*) The data records which matched with at least one subscription in the current subscription list S_{M_i}. The reason behind the design choice of criterion *a)* is that the data records which matched with one of the previous subscriptions at iteration $(t - 1)$, even though their data values have already been updated, are potentially and likely to be included in top-k answers in the current iteration t compared to the rest of the data records in the dataset D_t. In the case that data updates are not significant, the combination of the data records sent by both

Algorithm 1: Procedure for aggregator M_i

On receiving subscriptions from BS

Replace S_{M_i} with new subscriptions
Execute check()

On a new periodic update for iteration t

Execute check()

Function check()

$L_{M_i} \leftarrow \emptyset$
for $d_{j,t} \in D_{M_i,t}$ which is not yet sent to BS **do**
  if $d_{j,t-1}$ matches with $\exists s \in S_{M_i}$ in $(t-1)$ iteration **then**
    $L_{M_i} \leftarrow L_{M_i} \cup \{d_{j,t}\}$
  else if $\exists s_i \in S_{M_i}(f_{s_i}(d_{j,t}) \geq \theta_{s_i})$ – subscription matching **then**
    $L_{M_i} \leftarrow L_{M_i} \cup \{d_{j,t}\}$
Send L_{M_i} to BS

Algorithm 2: Procedure for the base station (BS)

On receiving new data records $\bigcup_{i=1}^{N_A} L_{M_i}$

Let R be a set of data records BS has received so far in iteration t (also include $\bigcup_{i=1}^{N_A} L_{M_i}$)
for $q \in Q$ **do**
  Execute top-k processing for the current $T_{q,t}$ using data records in R
  Update $\theta_{q,t} \leftarrow a_{q,k_q}$ (k-ranked data score)
  if $\theta_{q,t} < \theta_{q,t-1}$ **then**
    $B \leftarrow$ true
if $B =$ true **then**
  Construct and issue an updated set of subscriptions

criteria *a)* and *b)* together should be sufficient for answering all top-k queries upon BS received them.

On the BS's side, in each iteration, BS receives many data records from the aggregators ($\bigcup_{i=1}^{N_A} L_{M_i}$). Algorithm 2 expects that those data can sufficiently answer latest top-k answers for iteration t. We update the final top-k answers of each query in Q by using those data. However, $\bigcup_{i=1}^{N_A} L_{M_i}$ is possibly not sufficient to guarantee the correctness if it is detected that the new threshold is lower than the previous threshold which is used in its disseminated subscription. In that case, issuing a new set of current subscriptions is needed, and it triggers the procedure of aggregators when receiving new subscriptions from BS (lines 1–2, Algorithm 1), and more than one round of communications is required to get sufficient data records for answering top-k queries.

This algorithm guarantees that, in each iteration, the process can be finished within 2-round communications (2 times of subscription dissemination) and assures the completeness of final top-k answers (no false negatives).

Lemma 1. *The algorithm can return the sufficient top-k answers (no false negatives) for all queries in Q to BS within 2 times of subscription dissemination for each iteration.*

Proof. Let R and R' be the data records BS has received so far in the first and second times of subscription dissemination for iteration t respectively. We make a proof by contradiction. Assume that, the subscription constructed in the second round cannot request sufficient answers in response to query q. In other words, $T_{q,t} \not\subset R'$.

The algorithm guarantees that $|R| \geq k_{max}; k_{max} = \max_{q \in Q} k_q$ because it is designed to send at least the answers of the previous round which are used to satisfy the complete set of answers in the previous round. In the second time, the threshold $\theta_{q,t}$ of each query q is set as k-ranked data score in R.

Thus, $|\{d_{i,t} | d_{i,t} \in R \wedge f_q(d_{i,t}) \geq \theta_{q,t}\}| = k_q$. By issuing these subscriptions to every aggregator, the set $\{d_{i,t} | d_{i,t} \in D_t \wedge f_q(d_{i,t}) \geq \theta_{q,t}\}$ will be returned in R'. Due to the fact that $R \subset D_t$, it holds $|\{d_{i,t} | d_{i,t} \in D_t \wedge f_q(d_{i,t}) \geq \theta_{q,t}\}| \geq k_q$ as well as $\{d_{i,t} | d_{i,t} \in D_t \wedge f_q(d_{i,t}) \geq \theta_{q,t}\} \subset R'$. Therefore, by the definition of a top-k query, it shows that $T_{q,t} \subset \{d_{i,t} | d_{i,t} \in D_t \wedge f_q(d_{i,t}) \geq \theta_{q,t}\} \subset R'$ which contradicts our assumption. □

Nevertheless, this algorithm may incur some false positives which are data records

sent to BS but not belong to top-k answers. The set of false positives at iteration t is as defined in the following equation.

$$F_t = \bigcup_{i=1}^{N_A} L_{M_i} \setminus \bigcup_{q \in Q} T_{q,t} \tag{2.3}$$

We partially use an example of query q_1 in Fig.2.4 and consider 2 consecutive iterations, *i.e.*, iteration t and iteration $t+1$. Given that at iteration t, a subscription of q_1 matched $d_{1,t}, d_{2,t}, d_{5,t}$ and $d_{4,t}$ respectively resulting in $T_{q_1,t} = \{d_{1,t}, d_{2,t}, d_{5,t}, d_{4,t}\}$. Assume that at iteration $t+1$, this subscription matched $d_{1,t+1}, d_{2,t+1}, d_{5,t+1}$ and $d_{3,t+1}$. In this case, $\bigcup_{i=1}^{N_A} L_{M_i}$ at iteration $t+1$ consists of 5 items including $d_{1,t+1}, d_{2,t+1}, d_{5,t+1}$, $d_{4,t+1}$ and $d_{3,t+1}$. If finally $T_{q_1,t+1} = \{d_{1,t+1}, d_{2,t+1}, d_{5,t+1}, d_{3,t+1}\}$, then we can calculate the $\theta_{q_1,t+1}$ (if changed) and $d_{4,t+1}$ is a false positive. The reason we allow some false positives is because of Lemma 1, *i.e.*, guaranteeing the completeness of the answers within 2 iterations. We will latter show in the experiments that the number of false positives incurred by our procedures is not significantly larger than the other filter-based method.

Apart from the overhead due to false positives, the cost of subscription and its maintenance cost can be expensive and non-trivial. In the case that global data distribution is not changed much or unchanged, the disseminated subscriptions can be used in long term. On the other hand, if data are dynamically changed, this cost must be concerned especially when dealing with many concurrent queries. This is because we have to construct and disseminate subscriptions for every single query in Q.

For this aim, we propose a solution to reduce the number of subscriptions to be disseminated. It is noted that some queries with different query preferences in terms of both scoring function f and value k can share common answers, or even subsume to each other (query subsumption). Therefore, the subscriptions of some queries can be neglected to be disseminated. The proposed method aims to find a minimal set of subscriptions while keeping the same idea and guaranteeing the accuracy of top-k answers.

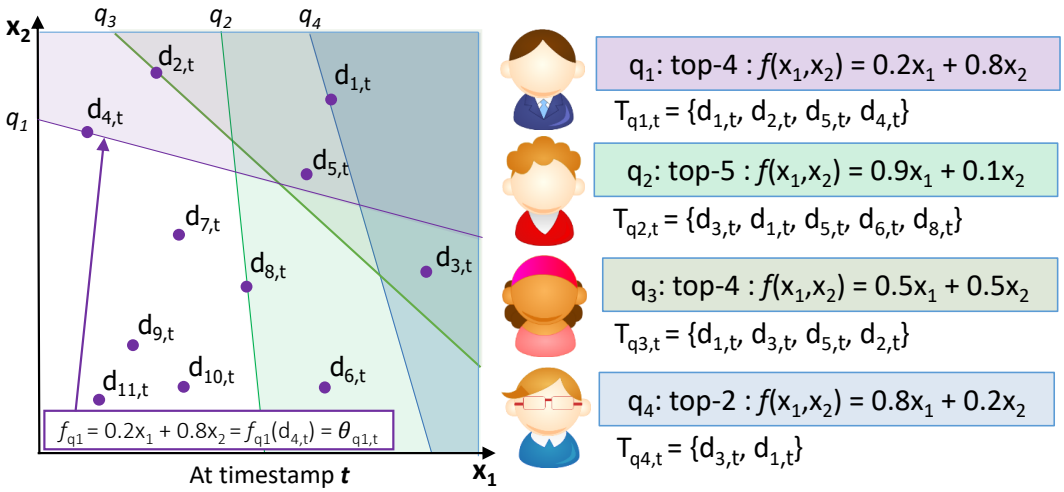


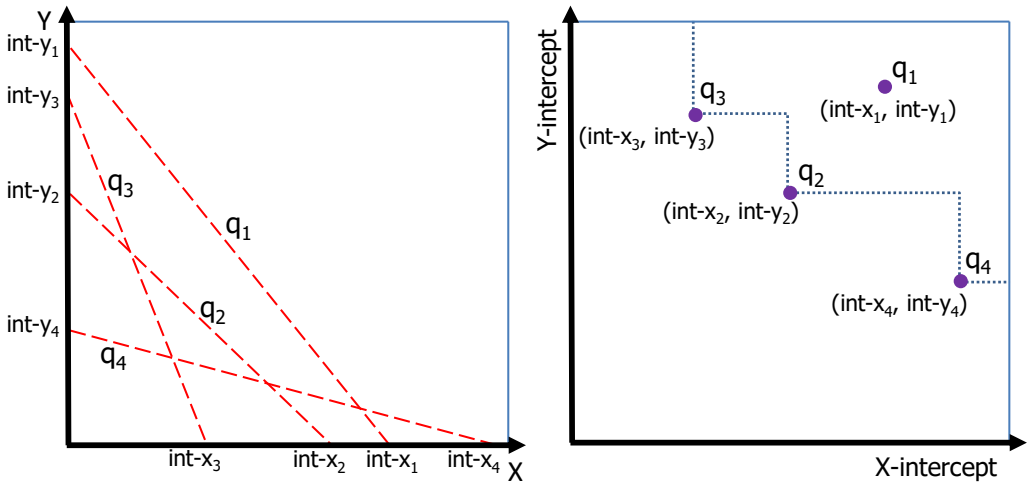
Figure 2.4: Example of 4 queries with different preferences having different affected areas and threshold lines

2.5 A Minimal Set of Subscriptions $S_{(M)}$

We try to find a minimal set of queries that the combination of these queries' answers can sufficiently satisfy any other queries in Q. A minimal set of subscriptions consists of the smallest set of subscriptions of queries which a part of their hyperplanes can be seen and not covered by other hyperplanes from the origin, for example, $\{q_1, q_2\}$ in Fig.2.4. The hyperplanes of the queries that are not able to be seen (ray shooting) from the origin, *i.e.*, subsumed queries, are unnecessary to be created and issued as top-k subscriptions. To issue only this minimal set can relieve the number of messages of issued subscriptions as well as messages for subscription updates that cost communication overhead. Therefore, how to compute and handle it efficiently when data are updated is an interesting issue, because a minimal set of subscriptions will dynamically go in and out. Moreover, it is desirable to support a lot of queries at a time (scalability) while keeping low computation cost.

2.5.1 Negative attribute values

The example as shown in Fig.2.4 as well as our following proposed techniques, only positive attribute values will be discussed. In the case of dealing with negative attribute values, translations of axes [72] must be applied to translate those negative values to



(a) The threshold lines of the 4 queries (b) The intercept space of the 4 queries

Figure 2.5: Example of representing 4 queries including q_1, q_2, q_3 and q_4

positive values.

Each data record $(d_{i,t}[1], d_{i,t}[2], \ldots, d_{i,t}[m])$ will be translated to $(d_{i,t}[1]+C, d_{i,t}[2]+C, \ldots, d_{i,t}[m]+C)$ in the translated m-dimensional Euclidean space where $C \in \mathbb{R}^{>0}$ and C must be large enough so that all attribute values in the current timestamp and latter timestamps are not negative after the translation, *i.e.*, $\forall d_{i,t} \in D_t, \forall j \in [1,m]$: $d_{i,t}[j] + C \geq 0$.

We use the new translated attribute values instead of original attribute values. Because this translation preserves the distances between every pair of data points, it will not affect the final results as well as their ranks in top-k queries, but only the raw score and the threshold have a surplus of a constant value compared with that of original Euclidean space.

2.5.2 A set of dominating subscriptions $S_{(D)}$

Obviously, the subscriptions of the queries whose top-k answers are fully included in other queries are not needed to be disseminated. In Fig.2.4, we can see that the answer set of query q_4 is a subset of the answer set of query q_2. To be precise, the answer space of query q_2 fully covers the answer space of query q_4. In this case, we can say that query q_2 dominates query q_4. Here, we call the set of queries that are not fully dominated by

any queries as dominating queries.

To identify a set of dominating subscriptions, we solve the axis intercepts of the linear equations (the scoring functions with their thresholds in Fig.2.5a), then those axis intercepts are plotted in the new space called *the intercept space* as shown Fig.2.5b. We define a set of dominating subscriptions $S_{(D)}$ as the queries lying on the skyline [10] (the lower the better) in the intercept space. Therefore, only acquiring the top-k answers of $S_{(D)}$ ($\{q_2, q_3, q_4\}$) also ensures the sufficiency of top-k answers for remaining queries $Q \backslash S_{(D)}$ ($\{q_1\}$) in Q.

Definition 2.5.1. (Dominating subscriptions) A set of dominating subscriptions $S_{(D)}$ contains all the queries that belong to the skyline in the intercept space.

As a result, unnecessary overhead can be saved, but in high dimensionality, it rarely occurs that queries are fully-covered to each other. Moreover, as in the given example in Fig.2.5a, query q_2 can also be eliminated because query q_3 and query q_4 together can subsume it. The following solutions in Section 2.5.3 and 2.5.4 try to eliminate that case.

2.5.3 Linear-optimization based method

Given a query list Q with their final answers, we can identify a set of minimal subscriptions $S_{(M)}$ by utilizing the linear optimization method in Eq.2.2 as shown in Algorithm 3. However, the computational complexity is very high because every query has to be tested repetitively. In practice, when there is an data update that affects the final answers and makes the thresholds of some queries change, all have to be re-computed. It is unacceptable when BS dealing with a numerous number of queries at a time.

2.5.4 Geometry-based method

Instead of using the linear optimization-based method, we adapted the geometric relation, a point-line duality [12], for identifying a set of minimal subscriptions.

In earlier works, a point-line duality has been used for efficient top-k and reverse top-k query processing such as mapping data points in the Euclidean space to lines in the dual space [17, 96]. The authors in [14] proposed a geometric unified framework to answer k-snippet, k-depth contour and reverse top-k queries by utilizing their proposed

Algorithm 3: Linear optimization-based $S_{(M)}$

Input: Query list Q

Output: A minimal set of subscriptions $S_{(M)}$

1: $S_{(M)} \leftarrow \emptyset$
2: **for** query q in Q **do**
3: $Q' \leftarrow Q \backslash \{q\}$
4: Maximize θ_m in Eq.2.2 by using Q' as constraints and f_q as an objective function
5: **if** $\theta_q < \theta_m$ **then**
6: $S_{(M)} \leftarrow S_{(M)} \cup \{q\}$

I/O efficient k-lower envelope algorithm. In [16], the authors used a duality transformation to produce an efficient indexing scheme for efficiently reporting vectors with large projection for a given query direction q. Our work transforms queries in Q into points similar to [16], but we use it to identify a smaller set of subscriptions.

Query's hyperplane and its duality

Point-line duality [12] is a transformation that maps lines and points between a primal plane (m-dimensional space) and a dual plane (another m-dimensional space). We denote this transformation using a asterisk (*) as a superscript as defined in Eq.2.4. A top-k subscription of query q forms a threshold line $\ell_q : f_q(x) = \theta_q$ which can be derived to a dual plane as in Eq.2.5 (let w_i be a query weighting of i-th dimension of query q).

$$\ell : x_m = -\sum_{i=1}^{m-1} a_i x_i - b \Leftrightarrow \ell^* = (a_1, a_2, \ldots, b) \tag{2.4}$$

$$\ell_q : x_m = -\sum_{i=1}^{m-1} \frac{w_i}{w_m} x_i + \frac{\theta_q}{w_m} \Leftrightarrow \ell_q^* = (\frac{w_1}{w_m}, \ldots, \frac{w_{m-1}}{w_m}, -\frac{\theta_q}{w_m}) \tag{2.5}$$

It is noted that, for a point in a dual plane ℓ_q^*, first $m-1$ coordinates purely represent the preferential functions which are fixed for a single query, but m-th coordinate depends on the threshold $\theta_{q,t}$, *i.e.*, the k-th ranked final answer $a_{q,k_q} \in T_{q,t}$, which can be dynamically changed when affected by data updates.

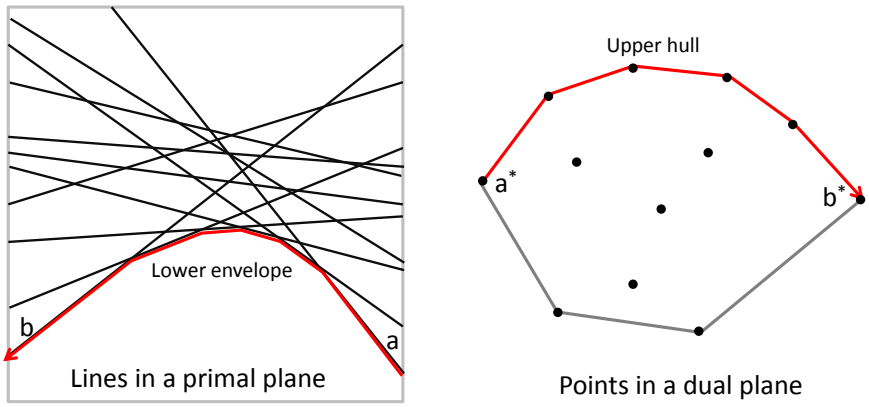


Figure 2.6: Lower envelope in a primal plane and upper hull in a dual plane

Lower envelopes of functions and convex hulls of their dual points

Consider a set of hyperplanes H in $\mathbb{R}^m$. The lower envelope of H in $(+x_m)$-direction is a set of all points that lie on the hyperplanes of H with the property that the vertical ray from those points along $(-x_m)$-direction does not intersect any of the hyperplanes in H. Here, we focus on finding which hyperplane participates in the lower envelope as an example of lower envelope in 2D shown in Fig.2.6. In the given example, there are 6 hyperplanes that participate in the lower hull with 5 intersection points and in the primal plane. It is noted that a part of the lower envelope will switch from one hyperplane to another according to 5 intersection points. With two non-parallel hyperplanes, there must be one intersection between them.

In geometry, hyperplanes of a lower envelope in a primal plane is an upper part of convex hull (upper hull) in a $+x_m$-directional dual plane. As illustrated in Fig.2.6, those 6 hyperplanes in the primal plane are equivalent to 6 points of the upper hull in the dual plane. Therefore, a method to identify the lower envelope is mapped to finding an upper hull in the dual plane. There are many efficient algorithms to find convex hull, *e.g.*, Quickhull [7] which also supports high dimensional data.

Identify a minimal set of subscriptions

We can see that finding a minimal set of subscriptions is strongly similar to finding a lower envelope, but finding the lower envelope of all queries in Q includes the lower envelopes outside positive planes (*e.g.*, the first quadrant in 2D) which are undesirable.

Here, we claim that finding a lower envelope of a set of dominating subscriptions $S_{(D)}$ instead of using all queries can resolve that problem, and the outcome is exactly what we want. This procedure is shown in Algorithm 4.

Lemma 2. *Only a minimal set of subscriptions will be the output of finding a lower envelope by using a set of dominating subscriptions* $S_{(D)}$.

Proof. Given an example in a 2 dimensional plane, only one possible way to disprove is that some hyperplanes in the lower envelope includes some outside the first quadrant Q_1. Therefore, there must be an intersection between two hyperplanes belonging to the lower envelope outside Q_1 because to generate a new segment of the envelope must have an intersection point. This leads to the contradiction because if the intersection occurs outside Q_1, then either of the corresponding two hyperplanes must not be included in a set of dominating subscriptions $S_{(D)}$ in the first place because either of them will be dominated. In addition, if a dominating subscription has an intersection with other dominating subscriptions inside Q_1 and it does not become a lower hull, it cannot belong to a set of minimal subscriptions definitely. □

Algorithm 4: Geometric-based $S_{(M)}$

Input: Query list Q

Output: A minimal set of subscriptions $S_{(M)}$

1: $S_{(D)} \leftarrow$ find_$S_{(D)}(Q)$ // described in Section 2.5.2

2: $S_{(M)} \leftarrow$ find_$S_{(M)}(S_{(D)})$ // described in Section 2.5.4

2.5.5 Updated procedure

We updated the algorithm for BS as explained in Algorithm 2 to be able to support the proposed idea of using a set of minimal subscriptions while the algorithm for aggregators remains unchanged. The new algorithm is shown in Algorithm 5.

Firstly, each query q must be attached with another threshold value apart from $\theta_{q,t}$ called *lower-bound threshold* $\theta_{q,lb}$. This threshold represents the lowest score that the data requested by the current $S_{(M)}$ so far is sufficient to answer the query q. Therefore,

as long as $\theta_{q,t} \geq \theta_{q,lb}$ for all query $q \in Q$, BS does not need to reconstruct and disseminate a new set of $S_{(M)}$ (lines 4–5). We calculate this threshold every time when $S_{(M)}$ is updated by using linear optimization as shown in lines 9–10. It is noted that, in low dimensionality, the size of $S_{(M)}$ is comparatively small compared with that of Q. Therefore, the number of constraints in the linear programming will be reduced resulting in faster computation.

Algorithm 5: Updated procedure for the base station (BS)

On receiving new data records $\bigcup_{i=1}^{N_A} L_{M_i}$

1: **for** $q \in Q$ using data in $\bigcup_{i=1}^{N_A} L_{M_i}$ **do**
2: Execute top-k processing for the current $T_{q,t}$
3: Update $\theta_{q,t}$
4: **if** $\theta_{q,t} < \theta_{q,lb}$ **then**
5: $B \leftarrow true$
6: **if** $B = true$ **then**
7: $S_{(M)} \leftarrow$ find_minsub(Q) // Algorithm 4
8: Issue $S_{(M)}$ to aggregators
9: **for** $q \in Q$ using data in $\bigcup_{i=1}^{N_A} L_{M_i}$ **do**
10: $\theta_{q,lb} \leftarrow$ Maximize θ_m in Eq.2.2 using $S_{(M)}$ as constrains and f_q as an objective function

2.5.6 Running time

The running time of our proposed method for each iteration can be divided into 2 separate parts. Firstly, the procedure at aggregators checks whether there exists at least one subscription in a list of minimal subscriptions which matches for each local data record. This takes $O(|D_{M_i,t}||S_{M_i}|)$ time. Secondly, in the procedure at BS, top-k query processing can be naively computed for all queries in $O(|Q|(|\bigcup_{i=1}^{N_A} L_{M_i}| + k \log |\bigcup_{i=1}^{N_A} L_{M_i}|))$ time. However, some additional index structures can be improved this computation cost, *e.g.*, using the branch and bound algorithm in R-tree [73]. Identifying a set of minimal subscriptions (if necessary) takes $O(|Q|^2)$ time in the worst case for finding $S_{(D)}$ (skyline calculation) [26] and takes $O(|Q|^{\lfloor m/2 \rfloor} + |Q| \log |Q|)$ time

for finding $S_{(M)}$ [12]. In background, calculating a lower-bound threshold for each query calls a linear program with $|S_{(M)}| + m$ constraints and m variables which is in $O(m|S_{(M)}|^2)$ time [9].

2.5.7 Handling too many false positives

Due to data dynamicity, a set of subscriptions which is disseminated so far possibly guarantees that it does not cause any false negative, but it may cause too many false positives. It is a trade-off between paying cost of updating subscriptions frequently to reduce false positives and using old valid subscriptions that may cause the cost of sending many false positives. For this aim, we design heuristic rules to decide when a new set of subscriptions should be renewed.

The number of false positives at iteration t (F_t) is calculated by using Eq.2.3. As a consequence, the observation of this value is varied by time (time series data) and is possibly highly fluctuated. Considering average case of false positives is more suitable. Therefore, we single out the exponential moving average to be used for this aim. The average number of false positives at iteration t is calculated by following formula.

$$F_{avg,t} = \begin{cases} \alpha_f F_t + (1-\alpha_f) F_{avg,t-1} & \text{, if } t \geq 1. \\ F_1 & \text{, otherwise.} \end{cases} \tag{2.6}$$

where α_f is a smoothing factor and $0 < \alpha_f < 1$.

This is to prioritize the latest information than the past average. Therefore, the average number of false positives will be adaptive in response to new data changes.

There is a break-even point where the cost due to false positives is over the cost of renewing subscriptions. For this aim, BS decides to renew subscriptions at iteration t when the following criteria is satisfied.

$$((F_{avg,t})B_{data}) - \left(|S_{(M)}|B_{sub}N_A\right) > 0 \tag{2.7}$$

where B_{data}, $|S_{(M)}|$ and B_{sub} are the size of data record, the cardinality of the set of minimal subscriptions and the size of a single subscription respectively.

2.6 Simulation Experiments

In this section, we setup a set of experiments to show the advantages of our proposed methods in each aspect by dividing into 3 subsections. In Section 2.6.1, we conduct experiments to evaluate our proposed algorithms in terms of execution time of identifying sets of subscriptions and their cardinality. In Section 2.6.2, we shows the analysis of communication cost among each method. Finally, we setup experiments and compare the performance of our proposed method with the others in terms of computation cost in Section 2.6.3.

2.6.1 Proposed method analysis

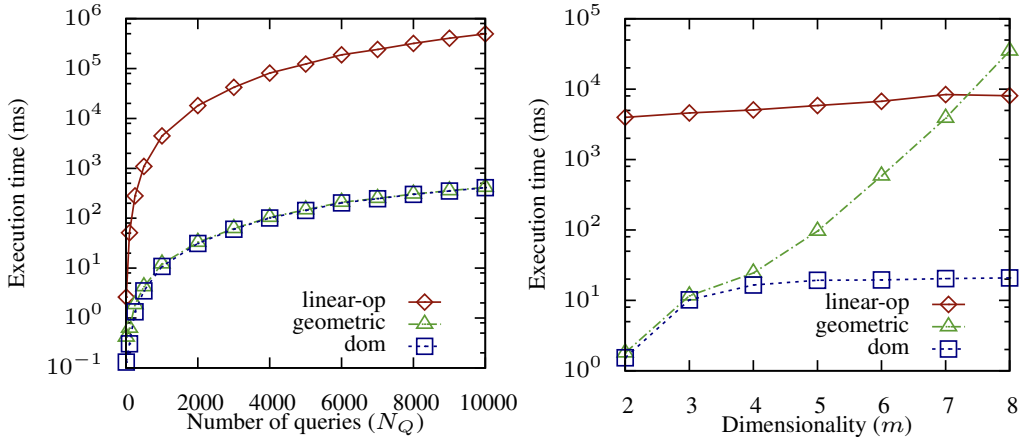
We preliminarily test the performance of our proposed technique to reduce the cardinality of the subscription set. Obviously, introducing more complicated algorithms yields more computation cost and latency as drawbacks. However, the aggregators' loads on receiving and transmitting data are largely reduced. In this experiment, we simulate a snapshot of the uniform data distribution in various dimensionality and various numbers of initial queries (N_Q). Query preferences are uniformly random and their value k is randomly drawn between 1 and k_{max}. We set the default parameters as follows, $m = 3$, $N_Q = 1000$ and $|D_t| = 500$. This setting is to show the scalability on high dimensionality and the number of queries.

Execution time

We compare the average execution time to identify a set of minimal subscriptions $S_{(M)}$ one time by using the linear optimization method and the proposed geometric-based method explained in Section 2.5.3 and 2.5.4 respectively implemented by using C# and the libraries[1,2].

Fig.2.7a shows the execution time in millisecond (log scale) on the impact of increasing number of queries when fixing dimensionality. The geometric-based method (geometric) runs faster than the linear optimization method (linear-op) significantly denoted by its growth rate. Therefore, the geometric-based method has a scalability on the

[1]Microsoft Solver Foundation: https://msdn.microsoft.com/en-us/library/ff524509(v=vs.93).aspx
[2]MIConvexHull: http://miconvexhull.codeplex.com/



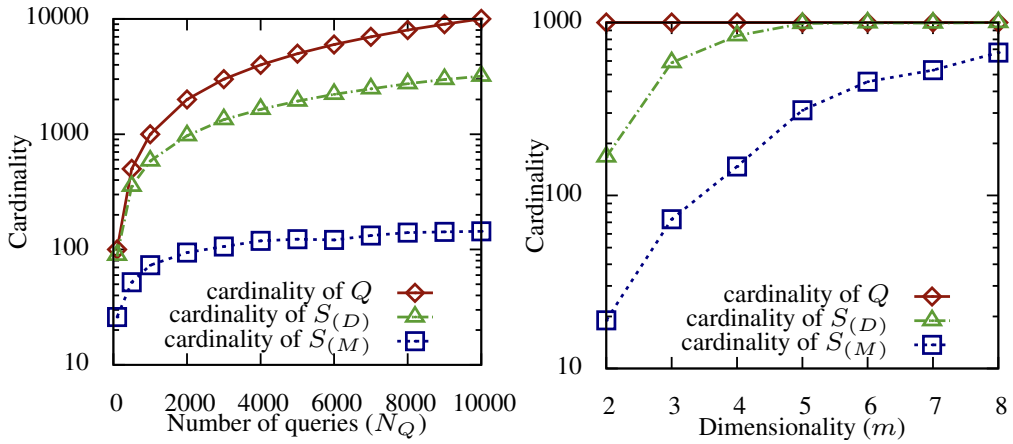
(a) Varying N_Q (b) Varying m

Figure 2.7: The execution time for identifying $S_{(D)}$ (dom) and $S_{(M)}$ using the linear optimization method (linear-op) and the geometric-based method (geometric)

number of queries. We also compare the execution time to identify $S_{(D)}$ (dom) which is skyline execution as shown in the figure.

In Fig.2.7b, we test the impact of increasing dimensionality with fixed number of queries. The geometric-based method runs faster than the linear optimization method in most cases. Due to high complexity of the convex hull in higher dimensionality, unlike Fig.2.7a, the computation cost of computing convex hulls increases rapidly compared with the cost of skyline computation. As a result, the geometric-based method spends more time than the linear optimization method when $m > 7$. Nevertheless, as surveyed, a large number of acquired real-world datasets in the related research in query processing as appeared in [6, 39, 58, 61, 87, 96, 100] mostly have dimensionality ranging between 2 to 5. It is noted that the execution time for identifying $S_{(D)}$ (dom) does not increase much and converges when $m > 3$.

Even though, in high dimensionality, the execution time for identifying $S_{(M)}$ is comparatively high, but the period length of update time depends on applications and data models, for example, weather information in the example aggregating data every 10 minutes or a smart meter which may probe the current resource usage every hour. The 2 times of execution time of identifying $S_{(M)}$ must be finished within the given period length of time. Therefore, we have to consider the proper dimensionality and the num-



(a) Varying N_Q (b) Varying m

Figure 2.8: Comparison of the sizes of query list Q, a set of dominating subscriptions $S_{(D)}$ and a set of minimal subscriptions $S_{(M)}$

ber of queries regarding to the available amount of computation time. It is noted that, if data in that iteration are unchanged or changed a little from the previous iteration, a procedure for identifying $S_{(M)}$ is possibly unnecessary to execute in that iteration.

Subscription reduction rate

Both linear optimization method (linear-op) and geometric-based method (geometric) give the same output. This time, we show how many subscriptions can be reduced compared with naively send all subscriptions in Fig.2.8. The vertical axis shows the number of subscriptions, and the horizontal axis represents the initial number of queries. In Fig.2.8a, increasing the number of queries with fixed dimensionality, the number of minimal subscriptions is very small compared with a set of full queries and dominant queries. In Fig.2.8b, as dimensionality increases while fixing the number of queries, the cardinality of minimal subscriptions $S_{(M)}$ and the cardinality of dominating subscriptions $S_{(D)}$ also increase significantly because the cardinality of skyline, and the cardinality of a lower hull increases exponentially and converges to the sizes of Q and $S_{(D)}$.

Nevertheless, at $m = 8$, the cardinality of $S_{(M)}$ is still lower than the full set of queries while only a set of dominant queries itself is very close to entire queries since

Table 2.1: Simulation parameters for Section 2.6.2

Parameter	Default	Range
k_{max}	20	5–60
N_D in the SYN dataset	1000	200–4000
N_Q	250	10–10000
N_A	4	-
Dimensionality m	3	2–8
No. of iterations in the SYN dataset	2000	-
α_f (Eq.2.6)	0.1	-
p (Eq.2.8)	0.3	-

$m = 5$. Therefore, identifying $S_{(M)}$ can reduce the number of subscriptions to be forwarded to aggregators numerously when there are a large number of queries and especially in low dimensionality.

2.6.2 Communication cost analysis

Experiment setup

Some experiments are conducted by using an event-based simulator implemented in Java for measuring communication cost. We assumed that the coordinator server (BS) can directly communicate with all aggregators. The range and default setting of the parameters are expressed in Table 2.1. Each aggregator M_i equally holds $|D_i| = N_D/N_A$ data records. Firstly we initially inject N_Q 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 periodically occurring in the system as events.

Measurement

We record 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 is counted by how many data records and subscriptions transmitted between BS and aggregators. We define the size of floating point and integer equal to 8 bytes and 4 bytes respectively. Therefore, a top-k subscription consisting of id (integer), a scoring function (m floating points), value k (integer) and a threshold (floating point) is $(8(m+1)+4(2))$ bytes while a data record consisting of id (integer), timestamp (integer) and data values is $(8m+4(2))$ bytes.

Datasets

In this experiment, we use both synthetic and real datasets to simulate and show our proposed method's performance.

1. **Synthetic dataset (SYN):** Firstly, each data record $d_{i,1}$ is uniformly random on each dimension as a point on the m-dimensional data space. In the real practice, data at each epoch $d_{i,t}$, are likely to be changed a little bit from its previous value $d_{i,t-1}$ or unchanged. We model a data value on each dimension as a Gaussian random walk pattern following $d_{i,t}[j] = d_{i,t-1}[j] + \lambda_{i,t} e_t[j]$ where $e_t[j] \sim \mathcal{N}(0, 0.1)$ (normal distribution), $1 \leq j \leq m$ and

$$\lambda_{i,t} = \begin{cases} 1 & \text{, with probability } p. \\ 0 & \text{, with probability } 1-p. \end{cases} \tag{2.8}$$

2. **Weathernews (WN):** Weathernews consists of the history of weather information which is recorded every 10 minutes and aggregated from 821 weather stations across Japan. We chose 3 attributes to be used including, temperature, wind speed and precipitation of 10 minutes. Each aggregator holds a set of data records equally split from 821 data records of weather information. We simulate the system by using this dataset for 1 week which consists of 2020 iterations of updates. This dataset represents the characteristics of high dynamic data changes especially in some attributes, *e.g.*, wind speed.

Comparison methods

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

1. **Centralized method (CEN):** The baseline which all data records and information of updates are sent to BS regardless of queries.

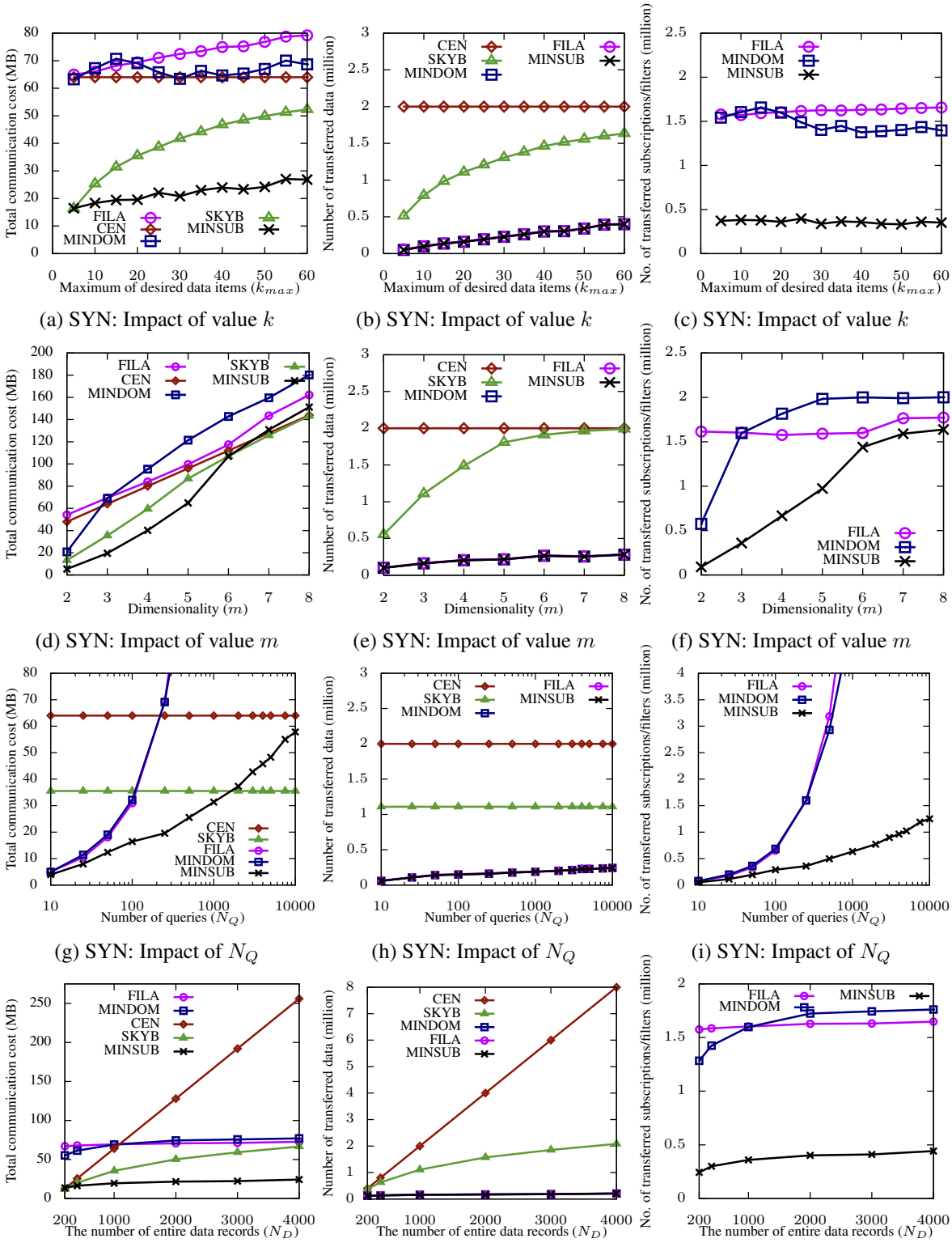
2. **K-skyband (SKYB):** The data records which belong to K-skyband are sufficiently enough for answering any top-k queries where $k \leq K$. BS simply aggregates K-skyband from every local node, so any top-k queries at BS can be answered intermediately by the aggregated data records. Due to the data 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 records D_i, *i.e.*, transferring all data records like the CEN method.

3. **FILA [91]:** We implemented this method by constructing filters for each individual query (N_Q filters in total) and applied them to every aggregators. For a fair comparison, the eager-update uniform filters for order-insensitive top-k monitoring were used. In each iteration, the data records which do not pass the filter will not be sent to BS. BS needs to probe some additional data records and update the filter bounds if those values have changed. Hence, the cost of filter updates and probes cannot be neglected. In the experiment, we include the number of probes in the number of filter updates due to the same message size.

4. **MINDOM:** Our proposed method described in Section 2.5.5 by disseminating dominating subscriptions $S_{(D)}$ as a list of subscriptions instead of minimal subscriptions $S_{(M)}$ (note that $S_{(M)} \subseteq S_{(D)}$).

5. **MINSUB:** Our proposed method described in Section 2.5.5 by disseminating minimal subscriptions $S_{(M)}$.

We also have tested the performance of naively disseminating subscriptions described in Section 2.4.1, but the performance is worse than others significantly.



(a) SYN: Impact of value k (b) SYN: Impact of value k (c) SYN: Impact of value k

(d) SYN: Impact of value m (e) SYN: Impact of value m (f) SYN: Impact of value m

(g) SYN: Impact of N_Q (h) SYN: Impact of N_Q (i) SYN: Impact of N_Q

(j) SYN: Impact of N_D (k) SYN: Impact of N_D (l) SYN: Impact of N_D

Figure 2.9: Results from the synthetic dataset (SYN) showing the total communication cost in the first column, the number of transferred data in the second column and the number of transferred subscriptions/filters in the third column

Results of the synthetic dataset (SYN)

Impact of the number of desired data records k_{max} The results of the total communication cost, the total number of sent data records and the total number of transferred subscriptions or filter updates are shown in Fig.2.9a, Fig.2.9b and Fig.2.9c respectively. In Fig.2.9a, the total communication cost of the CEN method is constant regardless of k_{max} because it sends all data records in every epoch to BS. As k_{max} increases, the number of data records in K-skyband to be sent to BS increases dramatically (Fig.2.9b) and causes a huge communication cost.

Even though the number of transferred data records of the FILA, MINDOM and MINSUB methods also gets increasing with k_{max}, their increments are much slower than the SKYB method. The FILA method which maintains and keep updating many filters for every query as well as the MINDOM method which disseminates all dominating subscriptions if they have changed suffer from sending a massive amount of those messages compared with the MINSUB method shown in Fig.2.9c. This makes the total communication cost of the FILA and MINDOM methods worse than the CEN method for $k_{max} > 5$ while the MINSUB method outperforms the others. It is noted that, when k_{max} is high, the number of sent data records increases certainly, but this reduces the chance that the previous disseminated subscriptions become invalid. Therefore, the number of times of subscription reconstruction and dissemination becomes lower resulting in a slight decrease of total number of sent subscriptions as shown in Fig.2.9c.

Impact of dimensionality m In Fig.2.9d, the total communication cost of all methods surges dramatically with dimensionality. The communication cost of the CEN method increases linearly due to the larger size of data records. The cardinality of K-skyband also increases numerously. At $m > 6$, the cardinality of K-skyband is almost the same as the size of the entire dataset. Therefore, the number of transferred data records shown in Fig.2.9e of the SKYB method increases significantly while this number that of the FILA, MINDOM and MINSUB methods slightly increases.

In Fig.2.9f exhibits more details about the huge cost of the FILA, MINDOM and MINSUB methods. The number of transferred subscriptions grows significantly especially in the MINDOM method because of the less chance of query domination in high

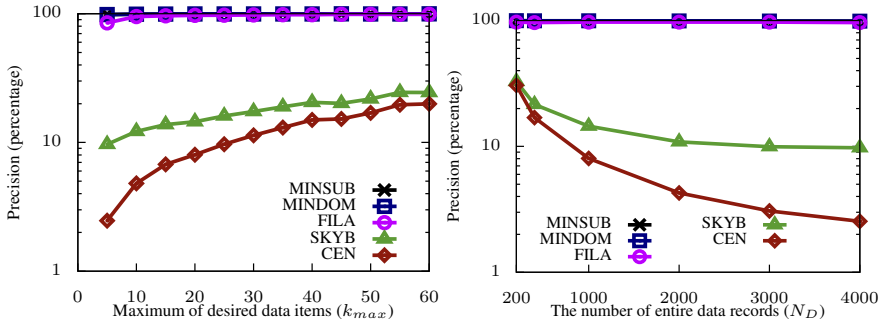
dimensionality. Disseminating only a set of minimal subscriptions can relieve this overhead, but its cardinality tends to increase with dimensionality. Though the number of filter updates of the FILA method is not affected much by the dimensionality unlike the MINDOM and MINSUB methods, this number is high due to the usage of multiple filters for multiple continuous queries. As a result, when $m > 3$, the FILA method performs the worst whereas the MINSUB method outperforms the others in low dimensionality ($m < 6$). Nonetheless, in the real practices, the number of dimensions, the number of queries or even the dynamicity of data changes are likely to be less than this experimental setup, The performance can vary based on the application. It is noted that the size of data records is set as small as the size of subscription, but in some applications data records are attached with other explanatory metadata. Due to the capability of the MINSUB method to keep small transferred data records, the MINSUB method is more beneficial than the other methods in the case that the size of a data record is larger than the size of a subscription.

Impact of the number of queries N_Q In this setting, we vary the number of continuous queries injected to the system at the initialization from 10 to 10000 (shown in a log scale). Only the FILA, MINDOM and MINSUB methods have an effect on this factor illustrated in Fig.2.9g. Though these methods can always keep the number of transferred data records low (Fig.2.9h), the number of transferred subscriptions or filter updates causes the huge sum of total communication cost (Fig.2.9i). It is noted that these numbers rise with N_Q. Therefore, the MINDOM and FILA methods cannot outperform the SKYB method if $N_Q > 100$ while the MINSUB method performs worse than the SKYB method when $N_Q > 2000$.

This can be a trade-off of the filter-based and subscription-based methods that in the case of large number of continuous queries, not only the communication cost but also the computation cost to create the subscriptions or to maintain valid filters are possibly higher than the naive method.

It is noted that the number of combinations of preferences can be limited, for example, when users adjust the weightings via questionnaires. Those users who share the same preference can share the final result of a single query with regard to that preference without issuing a query for each user.

In addition, in some cases such as correlated data, the top-k results of various queries



(a) SYN: Impact of value k (b) SYN: Impact of value N_D

Figure 2.10: Precision evaluation showing the outperformed MINSUB and MINDOM methods and other 3 comparison methods using the synthetic dataset (SYN)

are likely to share the same set of ranked results. Users should be advised to pose queries in the system because some queries are possibly semantically redundant. To be able to avoid this redundancy, we can further avoid unnecessary computational and communication costs for a large number of queries.

Impact of the number of data records N_D In this setting, we vary the number of data records in each aggregator ($|D_{M_i,t}| = N_D/4$), and the results are presented in Fig.2.9j, Fig.2.9k and Fig.2.9l. In Fig.2.9j, the total communication cost of the CEN method goes up linearly as increasing the number of data records in the system while the total communication cost of the SKYB method gradually increases and converges. Still, the MINSUB method is outstanding from the other methods because its total communication cost remains almost constant regardless of N_D. Fig.2.9k shows that the number of sent data records of the FILA, MINDOM and MINSUB methods is nearly constant. This is because these methods issue the set of subscriptions/filters which denotes and customizes the space of users' interest in order to answer the queries individually. In contrast to the SKYB method, it greedily aggregates all data records by considering the dominant property, but this can cause more false positives to be returned than the MINSUB method. However, the number of transferred subscriptions or filters of the FILA and MINDOM methods shown in Fig.2.9l is obviously high. The MINSUB method can reduce this cost around 70% from the FILA method. Hence, the result stresses the strength of the MINSUB method on data scalability.

The number of false positives The filter-based and subscription-based methods, *i.e.*, FILA, MINDOM and MINSUB, were designed to prevent unnecessary data records (false positive data, FP). In this setting, we aim to evaluate the number of false positive data of each method by evaluating their precision (Eq.2.9).

$$precision = \frac{TP}{\#data} = \frac{TP}{TP + FP} \tag{2.9}$$

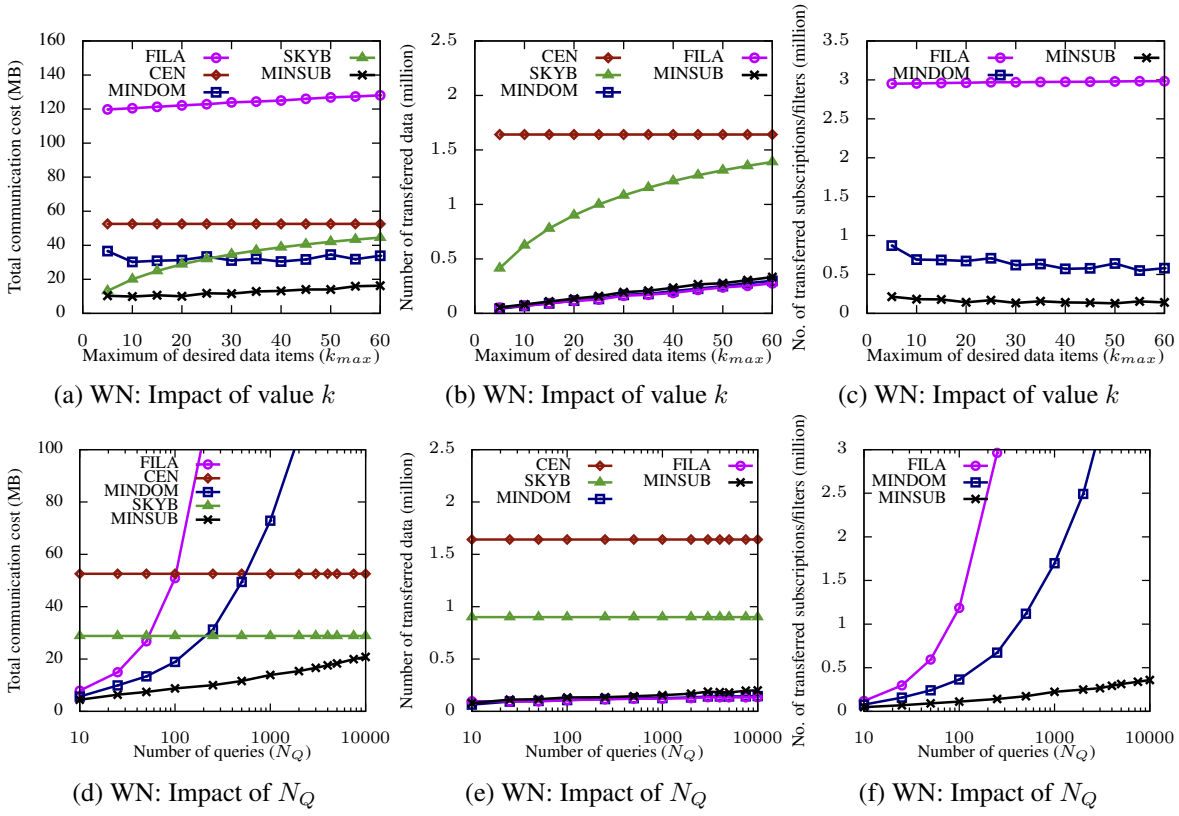
where TP is the total number of actual top-k answers and $\#data$ is the total number of received data records at BS. Therefore, the higher the precision, the lower the number of false positives.

Showing the impacts of only two parameters in Fig.2.10, the overall precisions of the SKYB method and the CEN method get better when increasing k, but they get lower when increasing N_D. Even though our proposed method possibly incurs false positives, we found that the overall precision of the MINSUB method is kept high around 98% in all experiments unlike the CEN and SKYB methods whose precisions are below 25%.

Comparing the results in Fig.2.10 with Fig.2.9b and Fig.2.9k respectively, we can see that most of data records sent by the SKYB method and the CEN method are wastefully transferred while paying the cost of setting filters and subscriptions can successfully prevents sending such undesired data records. In addition, the MINSUB method can further cut that cost resulting in saving more total communication cost.

Results of Weathernews dataset (WN)

Impact of the number of desired data records k_{max} The results of the total communication cost, total number of sent data records and total number of transferred subscriptions or filter updates are shown in Fig.2.11a, Fig.2.11b and Fig.2.11c respectively. In this dataset, each data record in the Weathernews dataset is periodic weather information from each location. Therefore, the data values of some attributes in the consecutive timestamp can be sharply changed which is different from Gaussian random walk in the SYN dataset. Unlike the SYN dataset, the total communication cost of the FILA method is distinctively higher than the rest. This is because, in this dataset, the FILA method sends a lot of probe messages counted as filter updates shown in Fig.2.11c to ensure the correctness of the answers explained in [91].



(a) WN: Impact of value k (b) WN: Impact of value k (c) WN: Impact of value k

(d) WN: Impact of N_Q (e) WN: Impact of N_Q (f) WN: Impact of N_Q

Figure 2.11: Results from Weathernews dataset (WN) showing the total communication cost in the first column, the number of transferred data in the second column and the number of transferred subscriptions/filters in the third column

In the MINSUB method, it has benefits from receiving some data updates which used to be included in the subscription of the previous iteration (lines 6–7, Algorithm 1). Together with the capability to identify a smaller number of minimal subscriptions and the sufficiency of data records in BS for answering top-k queries, these can usefully avoid the necessity of the second round communication efficiently unlike issuing probe messages in the FILA method. As a result, the MINSUB method is able to save the total communication cost by keeping the number of transferred data records and subscriptions low.

Impact of the number of queries N_Q We vary the number of continuous queries injected to the system at the initialization from 10 to 10000. In this real dataset, even

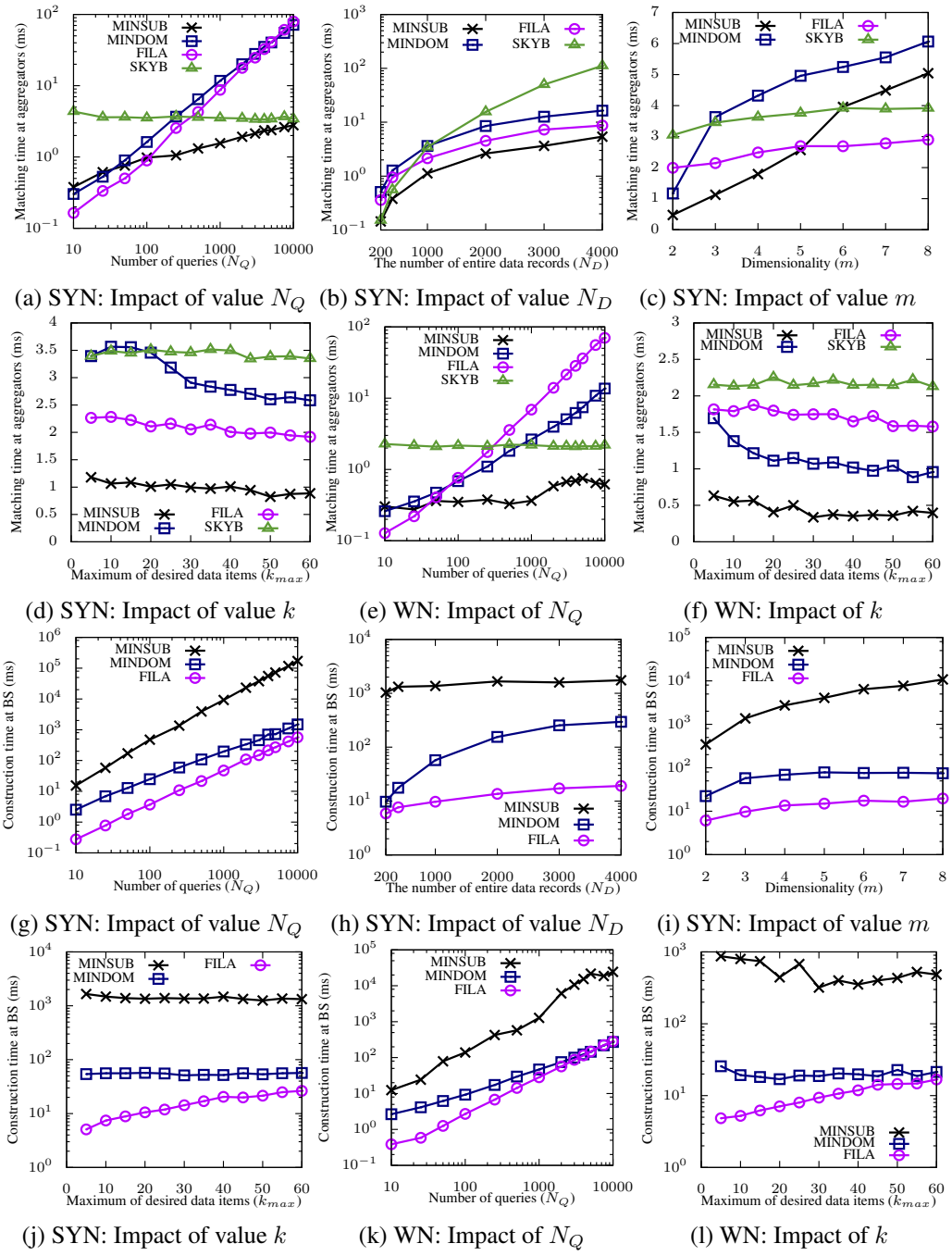
we increase N_Q to the maximum value at 10000, the total communication cost of the MINSUB method is still lower than that of the SKYB method as shown in Fig.2.11d. For the same reason, Fig.2.11e shows that the number of transferred data records is always kept low for the subscription-based and filter-based methods while the number of sent subscriptions of the MINSUB method is very small compared with the other methods in Fig.2.11f.

2.6.3 Computation cost analysis

In this section, we evaluate the main computation cost incurred at the aggregators which are assumed to be much less powerful than the single coordinator server (BS) in terms of computational capability and power resources (*e.g.*, battery-powered sensor nodes) and also show the main computation cost at BS.

Apart from the communication cost, the aggregators need to forward the data records that are matched to the subscriptions or filters. Therefore, the computation cost at the aggregators is represented by the total subscription/filter matching time at aggregators for the MINSUB, MINDOM and FILA methods and by K-skyband calculation time for the SKYB method for each iteration. In the MINSUB, MINDOM and FILA methods, BS has to iteratively construct a set of filters/subscriptions considered as main computation cost at BS. The same environment as in Section 2.6.2 including default parameters, implemented methods and datasets is used in these experiments. We perform them in a virtual machine with 2GB and a single core of a 3.60GHz CPU. Because the computational time at both aggregators and BS can vary in each iteration depending on the data updates, parameters, etc., we report average computational time of all iterations.

Among 5 methods listed in Section 2.6.2, only the CEN method does not do any data processing at the aggregators. Also, only the CEN and SKYB methods do not incur any computation cost at BS for constructing a set of filters/subscriptions. Actually, the CEN and SKYB method have to perform top-k computation at BS. However, this computation time is comparatively small, so we neglect this cost and omit from the results.



(a) SYN: Impact of value N_Q (b) SYN: Impact of value N_D (c) SYN: Impact of value m

(d) SYN: Impact of value k (e) WN: Impact of N_Q (f) WN: Impact of k

(g) SYN: Impact of value N_Q (h) SYN: Impact of value N_D (i) SYN: Impact of value m

(j) SYN: Impact of value k (k) WN: Impact of N_Q (l) WN: Impact of k

Figure 2.12: Results of subscription/filter matching time at aggregators (a–f) and subscription/filter construction time at BS (g–l) at each iteration

Results of subscription/filter matching time at aggregators

Impact of the number of queries N_Q The computation cost at aggregators is directly affected by this factor for all methods because increasing N_Q feasibly yields the increment of the number of filters/subscriptions disseminated to each aggregator. At each iteration in the MINSUB, MINDOM and FILA methods, each aggregator has to verify which existing data record is matched to the stored subscriptions or filters. In the result of the SYN and WN datasets as shown in Fig.2.12a and Fig.2.12e respectively, even though the MINSUB method can largely reduce the number of subscriptions compared with the number of filters in the FILA method (Fig.2.9i), subscription matching in the MINSUB and MINDOM method checks more conditions (Algorithm 1 lines 6–9) than filtering of the FILA method. Therefore, for small N_Q, the FILA method takes less computation time than others. However, when N_Q increases, this computation time of the MINDOM and FILA methods grows rapidly with N_Q because of the higher number of subscriptions/filters in accordance with Fig.2.9i and Fig.2.11f. The SKYB method executes local K-skyband of a set of local data records regardless of N_Q, so the computation time varies at a small degree. When $N_Q > 500$, the computation time of both MINDOM and FILA methods surpasses that of the SKYB method in both datasets while, at very high $N_Q = 10000$, the computation cost of the MINSUB method just comes close to that of the SKYB method in the SYN dataset and still below in the WN dataset.

Impact of the number of data records N_D Increasing N_D certainly affects the computation cost at aggregators for every method especially in the SKYB method due to K-skyband executions. As shown in Fig.2.12b, while the subscription/filter matching time of the MINSUB, MINDOM and FILA methods grows linearly in linear scale with N_D, this computation time of the SKYB method grows faster than the others significantly. It is noted that the MINSUB method incurs the lowest computation cost in this result because of a smaller set of subscriptions.

Impact of dimensionality m As m increases, subscription/filter matching time also increases due to more data attribute comparisons. According to Fig.2.12c, the computation time for all methods gets higher as expected. Even though the MINSUB and

MINDOM methods can reduce the number of subscriptions to be less than N_Q unlike the FILA method, when $m \geq 5$, both MINSUB and MINDOM methods perform worse than the FILA method. This is because, as m gets higher, the subscription reduction rate gets lower. Additionally, subscription matching is more complex than filtering. As a result, the computation cost of the FILA method is cheaper than others.

Nevertheless, the communication cost of the FILA method is the highest. It is noted that the SKYB method outperforms the MINSUB and MINDOM methods in terms of computation cost and communication cost in very high dimensionality ($m \geq 7$).

Impact of the number of desired data records k_{max} The computation cost varies directly with the number of subscriptions/filters, and k_{max} has an effect on the number of subscriptions. According to Fig.2.9c, the number of transferred subscriptions of the MINSUB method gets smaller when $k_{max} > 20$. Fig.2.12f shows the concordant result for the SYN dataset. In the same way, the result of the WN dataset in Fig.2.12l also corresponds to the result of the number of transferred subscriptions in Fig.2.11c.

Results of subscription/filter construction time at BS

Most of processing time at BS is due to subscription/filter construction at each iteration. The MINSUB method takes a large amount of time to identify a set of minimal subscriptions discussed in Section 2.5.6 resulting in a smaller set of subscriptions and lower computation cost at aggregators which have limitations in terms of computing capability, network resources and possibly battery power.

In all experimental results in this section, the FILA method outperforms other methods because the filter construction is not complex unlike identifying $S_{(D)}$ and $S_{(M)}$ in the case of the MINSUB method and only $S_{(D)}$ in the case of the MINDOM method. All results in various parameters and datasets are shown in Fig.2.12g–Fig.2.12l.

In summary, the computation cost at BS is directly affected by N_Q and m as shown in Fig.2.12g, Fig.2.12k and Fig.2.12i because the computational complexity of subscription/filter construction depends on the number of queries and dimensionality. Increasing N_D also increases the probability of frequent changes of threshold and frequent subscription/filter updates resulting in the increment of the average construction time in Fig.2.12h. The impact of k_{max} affects the computation cost at BS for all methods

because the cost of calculating k-ranked score gets higher. However, in Fig.2.12j and Fig.2.12l, this issue can be obvious seen only in the FILA method because the y-axis has been plotted on a log-scale, and the computation cost of the MINSUB and MINDOM methods instead relies much on the cardinality of N_Q and $S_{(M)}$.

Although, the computation cost at the MINSUB method is more expensive than the others, the MINSUB method provides much lower communication cost and lower computation cost at aggregators which are more crucial. It is noted that subscription construction at BS is executed in the time-gap between 2 iterations. Therefore, the available time to calculate depends on the applications and use cases (*e.g.*, data update every 5 minutes, every 3 hours and every day). Nevertheless, the computational capability of BS can be easily enhanced to meet the requirements, *i.e.*, the number of queries, dimensionality, time-gap between 2 iterations and so forth.

2.7 Conclusions

In this chapter, we addressed the issue of preference top-k monitoring queries which many users declare their continuous top-k queries on a 2-tier distributed system consisting of a coordinator server as a base station (BS) and data aggregators. A set of dynamic data records are aggregated at each aggregator, and these data records are updated periodically. Because each user desires a different set of top-k answers, the basic scheme for this problem is to aggregate all data records and relay them to BS in every epoch. To suppress massive wasteful data transmission, the objective of this work is to lessen the total communication cost between BS and data aggregators.

The proposed method constructs and issues a set of subscriptions to represent the limited data space of users' interest, so the aggregators relay only potential top-k candidates back to BS. Without any techniques, BS needs to issue excessive subscriptions of all queries which incur additional non-trivial communication cost. Hence, we propose a method to identify a set of minimal subscriptions to prevent this case, and proof that the method can give the perfect accuracy on data aggregation. Furthermore, we also propose how efficiently manage and maintain subscriptions while optimizing the low communication cost.

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 more preferable to the comparative methods on various factors in most cases.

As mentioned in Chapter 1, save communication cost by using our proposed method can increase network lifetime in power constrained systems while our design helps them support more concurrent queries and alleviate data bottlenecks at aggregators resulting in longer network lifetime and more scalability in terms of the number of users.

Chapter 3

Candidate Pruning Techniques for Skyline Monitoring

3.1 Introduction

Recently, many query processing methods have been developed and gained a lot of attentions in database researches in order to deliver most satisfactory results to various classes of end-users. Considering the dominance relations among objects (the competitiveness of each object), skyline computation [10], which represents a result set in which each result item is not worse than others, is also one of popular queries so far. An example usage of this query is commonly referred to to multi-criteria decision making of hotel selection [10].

In this chapter, we focus on skyline monitoring queries which deliver up-to-date skyline answers over frequent update streams. We stress this problem on frequent update streams where a large portion of observing data objects change their values in each timestamp (interchangeably called a snapshot). The data streams like this can be often seen in many real-life situations, for example, environmental monitoring and stock market analysis.

Environmental monitoring: Data readings from distributed weather stations are retrieved periodically to the server. An expert may observe and prioritize the region which has a flood risk noticed by high precipitation rate and water level by calculating skyline of those regions. However, the reading data can be fluctuated by

time, weather conditions, errors of sensors and so forth; thus, stable data readings across time cannot be expected. The skyline of those data must be calculated and observed continuously.

Stock market analysis: An investor tries to find interesting defensive stocks, *i.e.*, not fluctuating with the market and increasing gains. He/she may be interested in a list of stocks performing this behavior for a long period of time (*e.g.*, last 200 days) from the historical stock prices. Considering a day-time-frame snapshot, normally many stocks are changed in statistics. This model also accounts for a task to analyze other data archives such as sport data archives or web archives where many properties are expected to be changed in a consecutive points of time.

In skyline monitoring, a single data update can totally change a final skyline set, so handling multiple data updates at a time is challenging. Without any technique, to guarantee the correctness, the new skyline set must be computed from the entire set of data objects.

In this chapter, we propose an efficient method based on the properties of a bounding box (a minimum bounding rectangle in the case of 2 dimensions). We utilize bounding boxes to capture and prune unnecessary data candidates as well as neglect no-effect data updates. Therefore, we can identify a smaller candidate set in skyline computation in consecutive data snapshots resulting in saving overall execution time.

In summary, the contributions of this chapter are as follows:

- We formulate the problem definition of skyline computation on frequent data update streams as well as illustrate example applications of this problem.
- We propose an efficient algorithm and index structures to identify a smaller set of data candidates before skyline calculation, and the cost of maintenance is paid in according with degree of data changes (pay as you go).
- We conduct some experiments in various settings by using both synthetic and real datasets to show that our proposed method can run faster than the baseline and the comparison methods.

The organization of this chapter is as follows. The related work is explained in Section 3.2. In Section 3.3, we introduce the assumptions of the data model and re-

lated background knowledge for this chapter. In Section 3.4, we describe our proposed method. The performance of our proposed method through experiments is shown and discussed in Section 3.5. Finally, this chapter is summarized in Section 3.6.

3.2 Related Work

Skyline computation in database research was firstly introduced in [10]. The authors proposed two skyline algorithms including BNL skyline and D&C skyline algorithms. After that, numerous research papers tried to enhance the performance by using more complicated index structures such as Branch-and-Bound skyline algorithm [52].

Apart from the traditional skyline processing in databases, skyline processing for distributed systems has been studied as described in the survey [29]. Moreover, many interesting variants of skyline processing methods have also been studied for example, reverse skyline query [21], fragmented skyline [53], subspace skyline [74], uncertain skyline [22] and interval skyline [38].

The papers in [13, 34, 44] studied skyline monitoring over moving objects. They assume a kinetic model of moving data objects aiming to find the skyline objects (static attributes) when some dynamic attributes (locations or query points) are movable. These are quite different from ours, and their techniques are not suitable to solve our context's problems because a kinetic model is not assumed.

Some existing works [31, 45, 49, 71, 93] proposed efficient methods to continuously calculate skyline results over data streams. These aim to efficiently monitor the latest skyline set in sliding windows where window-range, data arrival time and data expiration time are given. Some algorithms together with indexing techniques are used to deal with data insertion and deletion into the assumed sliding windows. For the case of data modification assumed in this chapter, a single data modification (a data update) can be taken as two consecutive operations - insertion then deletion. This can incur a very large cost because there can be multiple data updates at each timestamp.

[75] assumed a very close problem to ours according to its data model. This work tries to monitor the latest modification of the skyline set when each data object is updated by the information from update streams. Its main contribution relies on allocating data into grid cells and consider the dominance relations between those grids to prune

unnecessary candidates from skyline calculation. Therefore, our experiment also adopts this technique as well as the method in [49] as the representative of its group to compare to our proposed method.

3.3 Preliminaries

3.3.1 Data model and its update model

We assume that the system analyzes a skyline set over a fixed number of data objects. Each data object is comprised of m numerical values as attributes and data id i as an object's identifier. Data attributes of data object i at the initialization (the first snapshot in the historical archive, snapshot 0) are represented as a tuple $p_i^0 = (p_i^0[1], p_i^0[2], \ldots, p_i^0[m])$. Let N be the number of all data objects in the system, a set of all data objects at snapshot t is denoted by $P^t = \{p_1^t, p_2^t, \ldots, p_N^t\}$, where $t = \{0, 1, \ldots, T\}$, and T is the total number of snapshots in the archive.

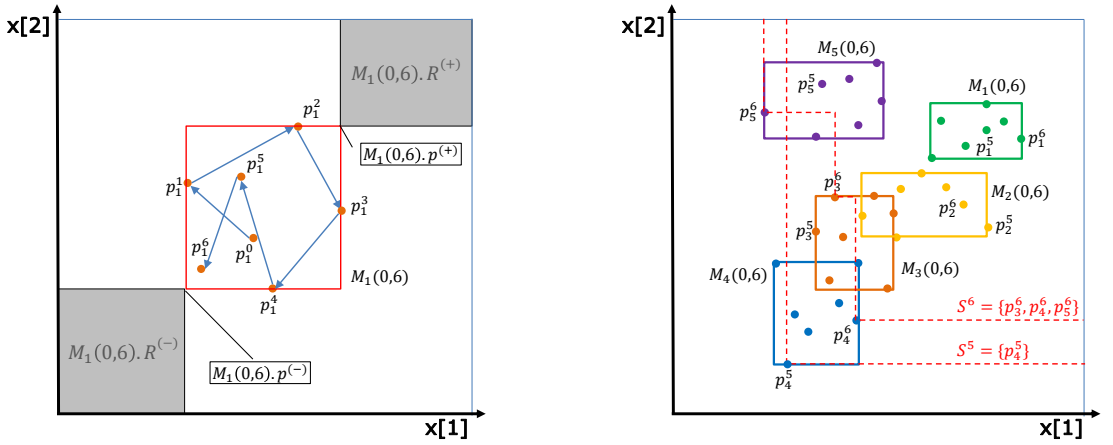
Definition 3.3.1. (Point dominance) A data point p_i^t dominates p_j^t ($p_i^t \prec p_j^t$) if and only if $\forall k \in \{1, 2, \ldots, m\} : p_i^t[k] \leq p_j^t[k]$, and $\exists l \in \{1, 2, \ldots, m\} : p_i^t[l] < p_j^t[l]$.

Definition 3.3.2. (Weakly point dominance) A data point p_i^t weakly dominates p_j^t ($p_i^t \preceq p_j^t$) if and only if $\forall k \in \{1, 2, \ldots, m\} : p_i^t[k] \leq p_j^t[k]$.

In this research, we assume that at each timestamp (snapshot) t, only a partial set of data objects changes their attributes' values from the previous timestamp $t - 1$. An update model like this can be often found in pull-based data delivery model that the server pulls new updates from data sources periodically.

How data change their values is described by an update tuple which can be defined in many ways based on applications, for example, a new value update defined by a 3-tuple $u = (i, t, (p[1], p[2], \ldots, p[m]))$ that means $p_i^t = (p[1], p[2], \ldots, p[m])$ and a modification update defined by a 3-tuple $u = (i, t, (\Delta p[1], \Delta p[2], \ldots, \Delta p[m]))$ that means $p_i^t = (p_i^{t-1}[1] \otimes \Delta p[1], p_i^{t-1}[2] \otimes \Delta p[2], \ldots, p_i^{t-1}[m] \otimes \Delta p[m])$ where $\otimes$ is an operator, such as addition, multiplication and average. This changes the corresponding data object p_i^{t-1} to p_i^t.

A list of updates of snapshot t (update streams) is a list of update tuples $U^t = \{u_1^t, u_2^t, \ldots\}$ where $|U| \leq N$. Therefore, data objects which are not modified by any



(a) An example MBR $M_1(0,6)$ summarizing $\{p_1^0, p_2^1, \ldots, p_1^6\}$

(b) An example of 2 skylines in snapshot $t = 5$ and $t = 6$

Figure 3.1: An example of MBRs and an example of skyline sets in the space

update tuples remain the same values that are $p_i^t = p_i^{t-1}$. Since our model embodies both multidimensional attributes (space) and time-series data (temporal data), it also works with spatio-temporal applications.

In this research, we aim to continuously calculate a set of skyline (S^t) efficiently from P^t at each consecutive snapshot t.

Definition 3.3.3. (Skyline set) Given a set of data points at snapshot t (P^t), $p_i^t \in P^t$ is included in the skyline set S^t if and only if $\forall p_j^t \in (P^t \backslash \{p_i^t\}), p_j^t$ does not dominate p_i^t ($p_j^t \not\prec p_i^t$).

3.3.2 Summarizing consecutive data snapshots with minimum bounding rectangles (MBRs)

A minimum bounding rectangle (MBR) is the smallest oriented rectangle enclosing a set of points which is a 2-dimensional case of a minimum bounding box in a coordinate system. Our proposed solution can deal with any number of dimensions by using the same idea. According to all examples in this chapter illustrated in a 2-dimensional space, for simplicity, we use the term *MBRs* to refer to this expression in general.

While each data object possibly changes its attributes' values at every timestamp t, those tracing data points can be seen as a set of points which can be summarized and represented as an MBR. Therefore, we use an MBR to summarize the space that a data object changes its values between consecutive snapshot a to b, *i.e.*, $\{p_i^a, p_i^{a+1}, \ldots, p_i^b\}$ where $a \leq b$.

An MBR of data object i from consecutive snapshot a to b is represented by a 3-tuple $M_i(a,b) = (i, p^{(+)}, p^{(-)})$ where $p^{(+)}[l] = \max_{j\in[a,b]} p_i^j[l]$ and $p^{(-)}[l] = \min_{j\in[a,b]} p_i^j[l]$ when $l \in \{1, 2, \ldots, m\}$. We represent a list of MBRs which ends at snapshot t as $M^t = \{M_1(a_1,t), M_2(a_2,t), \ldots, M_N(a_N,t)\}$ where a_i is a number of the start snapshot of MBR i. Fig.3.1a exemplifies 7 consecutive data snapshots of a data object ($i = 1$) denoted by $\{p_1^0, p_2^1, \ldots, p_1^6\}$ while the arrows express their trajectories between two consecutive data snapshots. Their MBR is a box $M_1(0,6)$ shown in the figure. For short, M_i^t refers to the latest MBR of object i at timestamp t regardless of the beginning timestamp ($M_i(*,t)$).

3.3.3 Dominance region and anti-dominance region

A dominance region of an MBR ($M_i^t.R^{(+)}$) is a subspace where $x[l] \geq M_i^t.p^{(+)}[l]$ for all $l \in \{1, 2, \ldots, m\}$. Any data points or MBRs that fully fall within this region will be weakly dominated by M_i^t. In the contrary, an anti-dominance region of MBR ($M_i^t.R^{(-)}$) is a subspace where $x[l] \leq M_i^t.p^{(-)}[l]$ for all $l \in \{1, 2, \ldots, m\}$. Any data points or MBRs that fully fall within this region weakly dominate M_i^t.

Definition 3.3.4. (MBR Dominance) An MBR M_i^t dominates M_j^t ($M_i^t \prec M_j^t$) if and only if $\forall l \in \{1, 2, \ldots, m\} : M_i^t.p^{(+)}[l] \leq M_j^t.p^{(-)}[l]$, *i.e.*, $M_i^t.p^{(+)} \preceq M_j^t.p^{(-)}$ (M_j^t fully falls in $M_i^t.R^{(+)}$).

Due to $M_i^t.p^{(-)}[l] = \min_{j\in[a,b]} p_i^j[l]$ when $l \in \{1, 2, \ldots, m\}$, we conclude that a point $M_i^t.p^{(-)}$ weakly dominates every point p_i^k where $k = \{a, a+1, \ldots, b\}$. We further define a definition of a set of skyline MBRs at snapshot t (S_M^t).

Definition 3.3.5. (Skyline MBR) Given a set of MBRs at snapshot t (M^t), $M_i^t \in M^t$ is included in the set of skyline MBRs S_M^t if and only if $\forall M_j^t \in (M^t \backslash \{M_i^t\}), M_j^t$ does not dominate M_i^t.

In addition, we denote a set of MBRs that do not belong to S_M^t as a set of non-skyline MBRs ($N_M^t = M^t \backslash S_M^t$).

Lemma 3. *$\forall M_i^t \in N_M^t$: there must be at least one MBR which is inside $M_i^t.R^{(-)}$.*

Proof. (Proof by contradiction) Assume that $M_i^t \in N_M^t$, but there is no MBR inside $M_i^t.R^{(-)}$. Due to $M_i^t \in N_M^t$ and Definition 3.3.5, there exists at least one MBR M_j^t dominating M_i^t. As a result, $M_j^t.p^{(+)}[l] \leq M_i^t.p^{(-)}[l]$ for $\forall l \in \{1, 2, \ldots, m\}$. From the explanation in Section 3.3.3, this leads to the contradiction because M_j^t must be contained in $M_i^t.R^{(-)}$. □

3.3.4 Pruning candidates for skyline calculation using MBRs

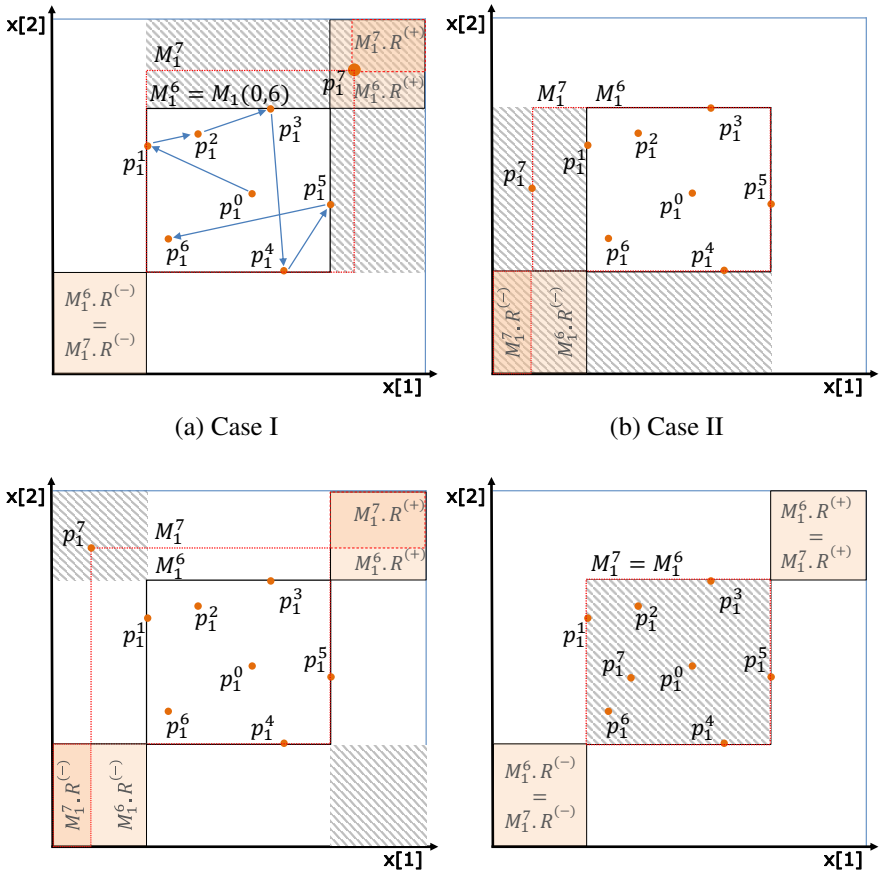
At each snapshot, instead of finding a skyline set from all data points P^t, we find the skyline by using only candidates in the skyline MBRs ($\{p_i^t | M_i^t \in S_M^t\}$). The cardinality of S_M^t is likely to be much smaller than that of P^t, so skyline calculation can be computed faster.

Lemma 4. *Skyline calculation from $\{p_i^t | M_i^t \in S_M^t\}$ produces the correct skyline set (S^t) as same as using all data points P^t.*

Proof. (Proof by contradiction) In order to produce incorrect S^t, there must be at least a point p_i^t where $M_i^t \in N_M^t \wedge p_i^t \in S^t$. By Lemma 3, there exists $M_j^t : M_j^t \prec M_i^t$. This leads to contradiction that $p_i^k \notin S^t$ because $\forall k : M_j^t.p^{(+)} \preceq p_i^k$. □

Running Example in Fig.3.1b

Fig.3.1b illustrates series of 7 data snapshots (data points) of 5 data objects ($i = \{1, 2, 3, 4, 5\}$) with their MBRs $(M_i(0, 6))$. By Definition 3.3.5, $M_1(0, 6)$ and $M_2(0, 6)$ are not skyline MBRs because they are dominated by $\{M_3(0, 6), M_4(0, 6)\}$ and $\{M_4(0, 6)\}$ respectively. Therefore, d_1^t and d_2^t are guaranteed not to include the final skyline at $t \in \{0, 1, \ldots, 6\}$ that we can safely remove them from skyline calculation. As in the example, consider only $t = \{5, 6\}$, while the final skyline at $t = 5$ (S^5) includes only $\{p_4^5\}$, S^6 includes $\{p_3^6, p_4^6, p_5^6\}$.



(a) Case I (b) Case II

(c) Case III (d) Case IV

Figure 3.2: MBR updates before including p_1^7 and after including p_1^7

3.3.5 Changes of MBRs when considering a new data snapshot

We start considering how a list of new updates at the next snapshot (U^{t+1}) affects the current MBRs M_i^t. Certainly, including a new snapshot of data can make M_i^t changed in size as well as their properties, *i.e.*, $p^{(+)}$, $p^{(-)}$, $R^{(+)}$ and $R^{(-)}$. Consider an MBR of data object i at snapshot t and its update tuple u_j^{t+1} where $u_j^{t+1}.i = i$ and $l \in \{1, 2, \ldots, m\}$, $M_i^{t+1}.p^{(+)}[l] = \max(M_i^t.p^{(+)}[l], p_i^{t+1}[l])$ and $M_i^{t+1}.p^{(-)}[l] = \min(M_i^t.p^{(-)}[l], p_i^{t+1}[l])$.

The effects of the data updates to M_i^{t+1} can be classified into 4 cases as follows:

(Case I) $M_i^{t+1}.p^{(-)} = M_i^t.p^{(-)}$ and $M_i^{t+1}.p^{(+)} \succ M_i^t.p^{(+)}$

This case happens when p_i^{t+1} falls in gray-shaded area at the right-top corner illustrated in Fig.3.2a. $M_i^{t+1}.R^{(-)}$ remains the same as M_i^t, but $M_i^{t+1}.R^{(+)}$ becomes

smaller.

(Case II) $M_i^{t+1}.p^{(-)} \prec M_i^t.p^{(-)}$ and $M_i^{t+1}.p^{(+)} = M_i^t.p^{(+)}$
This case happens when p_i^{t+1} falls in the gray-shaded area at the left-bottom corner illustrated in Fig.3.2b. In addition, $M_i^{t+1}.R^{(-)}$ becomes smaller than that of M_i^t while $M_i^{t+1}.R^{(+)}$ remains the same as M_i^t.

(Case III) $M_i^{t+1}.p^{(-)} \prec M_i^t.p^{(-)}$ and $M_i^{t+1}.p^{(+)} \succ M_i^t.p^{(+)}$
If p_i^{t+1} falls in the gray-shaded areas at the right-top and left-bottom corners illustrated in Fig.3.2c., this changes both lb and ub. In the same way, both $M_i^{t+1}.R^{(+)}$ and $M_i^{t+1}.R^{(-)}$ are degraded compared to that of M_i^t.

(Case IV) $M_i^{t+1}.p^{(-)} = M_i^t.p^{(-)}$ and $M_i^{t+1}.p^{(+)} = M_i^t.p^{(+)}$
M_i^{t+1} and M_i^t are identical if p_i^{t+1} falls inside M_i^t (gray-shaded area) illustrated in Fig.3.2d.

According to Definition 3.3.5, the membership of MBRs in S_M^t and N_M^t possibly no longer holds for snapshot $t+1$ due to the changes of its $R^{(+)}$ and $R^{(-)}$. Therefore, our proposed method introduces an efficient method to maintain the consistency in order to identify S_M^{t+1} as well as N_M^{t+1}. We describe details in the next section.

3.4 Proposed Algorithms

3.4.1 Overview

From the preliminaries in Section 3.3, we proved that only the data objects whose current MBR belongs to S_M^t are a sufficient candidate set for skyline calculation of each snapshot t. This can significantly reduce the cardinality of data candidates to be calculated in skyline computation. Therefore, we propose an efficient method to maintain those MBRs by keeping two separated lists including S_M^t and N_M^t where $M^t = S_M^t \cup N_M^t$ and $S_M^t \cap N_M^t = \phi$.

Our proposed method can be divided into 3 steps.

1. **Pre-computation maintenance (PRE)**
   According to new data updates from the stream at t, the MBRs at $t-1$ can be

Table 3.1: Notation Summary

Notation	Description
p_i^t	a data tuple of data object i at snapshot t
P^t	a set of data tuples at snapshot t
u_i^t	an i-th new update tuple at snapshot t
U^t	a set of new update tuples at snapshot t
$M_i(a, b)$	an MBR summarizing consecutive data snapshots between a and b of data object i
M_i^t, $M_i(*, t)$	refer to a current $M_i(a, b)$ where $b = t$ and a is neglected
S^t	a set of skyline points at snapshot t
S_M^t	a list of skyline MBRs at snapshot t
N_M^t	a list of non skyline MBRs at snapshot t

possibly changed in terms of physical MBRs and their pruning capability. This step tries to identify the correct S_M^t by paying low maintenance cost as much as necessary before skyline calculation.

2. **Skyline calculation (SKY)**
   This process is straight-forward. We calculate a final skyline result (S^t) by one of many state-of-the-art skyline computation methods but using a smaller set of candidates.

3. **Post-computation maintenance (POST)**
   Regarding to the final skyline result, we are able to detect some data objects whose MBR belongs to S_M^t but does not appear in S^t. In other words, these MBRs produce unpleasant false positives degrading overall the pruning capability. In this process, we propose a heuristic rule to solve this problem.

A brief pseudo-code of our proposed algorithm is shown in Algorithm 6. We explain the details of pre-computation maintenance (lines 1–9) in Section 3.4.3 and post-computation maintenance (lines 11–12) in Section 3.4.5.

Algorithm 6: Brief algorithm at snapshot $t > 0$

Data: The list of update tuples at snapshot t (U^t)
Result: The skyline set at snapshot t (S^t)

$V_L \leftarrow \emptyset$
foreach *Update tuple* $u_i^t \in U^t$ **do**
    Update M_i^t
    Add some MBRs needed to be further verified to V_L
foreach M_i^t *in* V_L **do**
    if *there exists* $M_j^t \in S_M^t : M_j^t \prec M_i^t$ **then**
        $N_M^t \leftarrow N_M^t \cup \{M_i^t\}$
    else
        $S_M^t \leftarrow S_M^t \cup \{M_i^t\}$
Calculate a skyline set S^t from $\{p_i^t | M_i^t \in S_M^t\}$
Check the false positives from S_M^t and S^t
Reconstruct MBRs in S_M^t that incur too many consecutive false positives

3.4.2 Intialization ($t = 0$)

At the initialization ($t = 0$), we have to construct the initial MBRs (M^0) of all data objects (P^0). That means, in each MBR M_i^0, $M_i^0.p^{(-)} = M_i^0.p^{(+)} = p_i^0$ for all $i \in \{1, 2, \ldots, N\}$. Hence, we calculate the first skyline set of P^0 and classify MBRs into 2 lists as $S_M^0 = \{M_i^0 | p_i^0 \in S^0\}$ and $N_M^0 = M^0 \backslash S_M^0$.

Moreover, we additionally introduce two important elements to help easily identifying the relations between MBRs in S_M^t and N_M^t including an i list of MBRs in a dominance region of each MBR ($M_i^t.M^{(+)}$) and a single i of MBR which is in an anti-dominance region ($M_i^t.d^{(-)}$). These relations can be established while executing skyline calculation by using the following rules:

1. If M_i^t is dominated by M_j^t, then $M_i^t \in N_M^t$, $M_i^t.d^{(-)} = j$ and $i \in M_j^t.M^{(+)}$.
2. If M_i^t is not dominated by any other MBRs, then $M_i^t \in S_M^t$ and $M_i^t.d^{(-)} = nil$ (not applicable).

According to Definition 3.3.5 and Lemma 3, we conclude that $\forall M_i^t \in N_M^t : M_i^t \neq nil$ while $\forall M_i^t \in S_M^t : M_i^t = nil$, and as long as M_j^t where $j = M_i^t.d^{(-)}$ dominates M_i^t, M_i^t must belong to N_M^t.

3.4.3 Data updates at snapshot $t > 0$

When receiving or considering a list of data updates $U^t = \{u_1^t, u_2^t, \ldots\}$, each update tuple describes how a new data tuple p^t can be generated while the rest (not indicated in U^t) $p_i^t = p_i^{t-1}$. For those unchanged data tuples, their MBRs are also unchanged, so the system does nothing in such a case.

However, we need to verify some affected MBRs both in S_M^{t-1} and N_M^{t-1} whether they are still in the correct lists at snapshot t due to new data updates. Therefore, we create an additional list called a verification list (V_L) containing MBRs whose elements are needed to be further investigated. This process included in the pre-calculation maintenance checks the following conditions.

1. $M_i^{t-1} \in N_M^{t-1}$ and M_i^t affected by p_i^t fall in either Case II or Case III (referred to Section 3.3.5).
   Because $M_i^t.R^{(-)}$ is deteriorating, $M_{M_i^t.d^{(-)}}^{t-1}$ may no longer dominate M_i^t. Therefore, we check if it still dominates, and if not, M_i^t is pushed to V_L.

2. $M_i^{t-1} \in S_M^{t-1}$ and M_i^t affected by p_i^t fall in either Case I or Case III (referred to Section 3.3.5).
   Because the dominance capability of M_i^t ($M_i^t.R^{(+)}$) has been reduced, some MBRs in $M_i^{t-1}.M^{(+)}$ may no longer be dominated by M_i^t. We move $j \in M_i^{t-1}.M^{(+)}$ which is not dominated by M_i^t to V_L.

3. Otherwise: No change of MBRs' status (neglected).

Verification of MBRs in V_L

We can see that all listed MBRs in V_L used to be in N_M^{t-1}, but possibly they are no longer able to be in N_M^t. This will increase the number of MBRs in S_M^t resulting in increasing in the number of candidates in skyline calculation. At this process, we try to check these MBRs again whether there exists at least an MBR in S_M^t dominating them before including them into S_M^t. Therefore, for each $M_i^t \in V_L$, we search for any first $M_j^t \in S_M^t$ that dominates M_i^t. If found, M_i^t is pushed back to N_M^t and set $M_i^t.d^{(-)} = j$. Otherwise, it is swapped to S_M^t.

It is noted that there may be more than one M_j^t dominating M_i^t. In best practice, M_j^t to be chosen should be an MBR in S_M^t which gives the longest distance between $M_i^t.p^{(-)}$ and $M_j^t.p^{(+)}$ ($M_i^t.d^{(-)} = \arg\max_{\forall j: M_j^t \prec M_i^t} d(M_i^t.p^{(-)}, M_j^t.p^{(+)})$). However, to do this approach consumes more time because we cannot avoid scanning the entire list of S_M^t. Heuristically, we may choose M_j^t having more pruning power than the others, *i.e.*, $M_j^t.p^{(+)}$ near the origin. Using the Manhattan distance, we choose $M_i^t.d^{(-)} = \arg\min_{\forall j: M_j^t \prec M_i^t} \sum_{l=1}^{m} M_j^t.p^{(+)}[l]$. Maintaining a list of S_M^t sorted by $\sum_{l=1}^{m} M_j^t.p^{(+)}[l]$, we can simply choose the first found dominating MBR in the list without scanning the entire list.

3.4.4 Skyline calculation

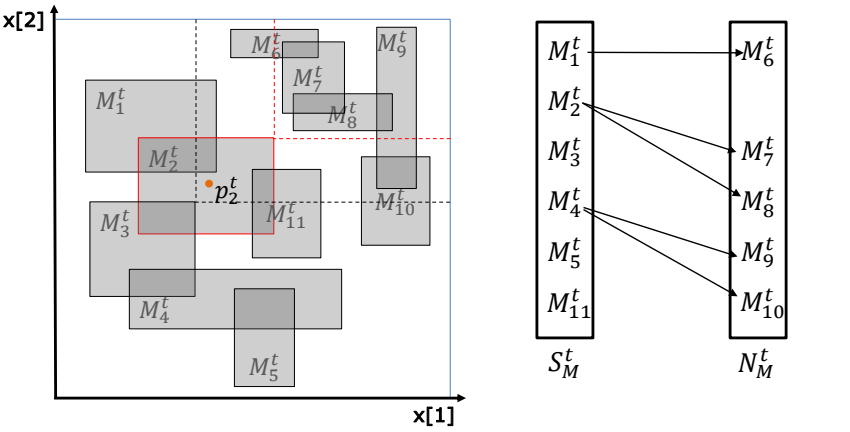
Because the main objective of this research aims to reduce the number of candidates to be calculated in skyline computation regardless of skyline computation algorithms. Therefore, we simply adopt the state-of-the-art Block-Nested-Loop skyline computation as default. According to Lemma 4, we calculate the final skyline set at snapshot t (S^t) by using a set of data points whose MBRs belong to S_M^t. Nevertheless, other complicated skyline computation algorithms can be used for further improvements, but this is out of the scope of this research.

3.4.5 Post-computation maintenance

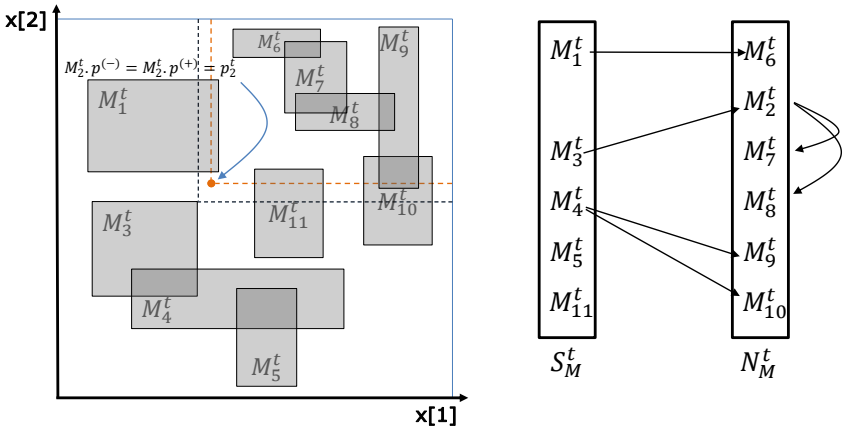
After S^t has been calculated, it is possible that some of candidates from S_M^t do not finally belong to S^t (false positives). If $M_i(*, t) = M_i^t$ usually incurs a false positive for a long period (too many consecutive snapshots), it is worth considering paying maintenance cost to reconstruct and newly start an MBR from the current snapshot t ($M_i^t = M_i(t, t)$) because of the possible higher gains of pruning capability in the next iteration.

Lemma 5. *The dominance and anti-dominance regions of a newly-reconstructed MBR $M_i' = M_i(t, t)$ are not smaller than the old $M_i = M_i^t(a, t)$ where $a < t$.*

Proof. The dominance region of M_i ($M_i.R^{(+)}$) is $x[l] \geq M_i.p^{(+)}[l]$ for $l \in \{1, 2, \ldots, m\}$. However, $M_i'.p^{(+)} = p_i^t$ and $p_i^t[l] \leq \max_{k \in [a,t]} p_i^k[l] = M'.p^{(+)}$. Hence, $M_i'.R^{(+)}$ must



(a) Before M_2^t reconstruction



(b) After M_2^t reconstruction

Figure 3.3: Running example of an MBR reconstruction illustrating that M_2^t is swapped to N_M^t after reconstruction

not be smaller than that of M_i. In the same way, the anti-dominance region of M_i ($M_i.R^{(-)}$) is $x[l] \leq M_i.p^{(-)}[l]$ for $l \in \{1, 2, \ldots, m\}$. However, $M_i'.p^{(-)} = p_i^t$ and $p_i^t[l] \geq \min_{k \in [a,t]} p_i^k[l] = M'.p^{(-)}$. Hence, $M_i'.R^{(-)}$ must not be smaller than that of M_i. □

MBR reconstruction strategy

MBR M_i^t that belongs to S_M^t but p_i^t is not included in S^t for a long period should be lowered the rank to N_M^t to decrease the cardinality of skyline calculation in each

snapshot. In this section, we discuss about a heuristic rule to decide which MBR should be reconstructed followed by a running example.

Firstly, a record of the number of consecutive false positives of each MBR should be tracked by adding a new MBR attribute called $M_i^t.f_p$. At each iteration, we calculate this parameter for all MBRs in S_M^t as follows:

$$M_i^t.f_p = \begin{cases} M_i^{t-1}.f_p + 1 & ; M_i^t \in S_M^t \wedge p_i^t \notin S^t \\ 0 & ; otherwise \end{cases} \tag{3.1}$$

We decide to reconstruct an MBR M_i^t when

$$M_i^t.f_p \geq \theta_f \tag{3.2}$$

where θ_f is a false positive tolerance threshold, *i.e.*, $M_i^t : \forall k \in [t - \theta_f, t] : M_i^k \in S_M^k \wedge p_i^k \notin S^k$.

Lemma 6. *If M_i^t is reconstructed, a list of MBRs M_i^t dominates ($M_i^t.M^{(+)}$) remains the same.*

Proof. Due to Lemma 5, the dominance region of M_i^t does not become smaller. Therefore, all MBRs that M_i^t dominated before reconstruction are still dominated by M_i^t after the reconstruction. □

Lemma 7. *After $M_i^t \in S_M^t$ is reconstructed, M_i^t may change its membership to N_M^t*

Proof. According to Lemma 5, the anti-dominance region of M_i^t may become larger. Therefore, it is possible that some $M_j^t \in S_M^t$ can dominate M_i^t. □

Running example Fig.3.3 illustrates an example of an MBR reconstruction. In Fig 3.3a, there are 11 different MBRs in the space which can be classified to S_M^t and N_M^t. An arrow from M_i^t to M_j^t shows the relation that M_i^t dominates M_j^t, *i.e.*, $M_j^t \in M_i^t.M^{(+)}$ and $M_j^t.d^{(-)} = M_i^t$. Assume that M_2^t is decided to be reconstructed at snapshot t and the recent data point p_2^t is as shown in Fig.3.3a. After M_2^t's reconstruction (Fig.3.3b), both dominance and anti-dominance region of M_2^t have changed, and M_2^t is no longer in S_M^t because it is dominated by M_3^t while $M_2^t.M^{(+)}$ ($\{7, 8\}$) remains the same (no additional cost of finding). Note that $M_i^t \in N_M^t$ can be dominated by some $M_j^t \in N_M^t$ (not only limited to $M_j^t \in S_M^t$).

In summary, while the process in pre-calculation maintenance swaps some MBRs from N_M^t to S_M^t, the process in post-calculation maintenance dynamically swaps back some MBRs from S_M^t to N_M^t. This responses to behaviors of data movement in an adaptive way.

3.5 Simulation Experiments

In this section, we conducted some experiments by implementing algorithms using C# on a single commodity PC. We tested our proposed algorithm as well as other competitive methods on the same environment. The system was to process a large file dataset which contains a series of $(D^0, U^1, U^2, \ldots, U^T)$ and find the desired output which is $(S^0, S^1, \ldots S^T)$. We evaluated the performance by measuring the wall clock time of total execution time in each method. In the results, only synthetic datasets were generated 3 times in each case, and the results report the average case of them.

3.5.1 Datasets

In this experiment, we used both synthetic and real datasets to simulate and show our proposed method's performance.

1. **Synthetic dataset (SYN):** Firstly, each data record p_i^0 is uniformly random on each dimension as a point on the m-dimensional data space $[0, 100]^m$. We model a data value on each dimension as a Gaussian random walk pattern following $p_i^t[l] = p_i^{t-1}[l] + u_i^t[l]$ where $u_i^t[l] = \lambda_i^t \cdot e_t[l]$, $e_t[l] \sim \mathcal{N}(0, 0.5)$ (normal distribution), $1 \leq l \leq m$ and
$$\lambda_i^t = \begin{cases} 1 & \text{, with probability } p. \\ 0 & \text{, with probability } 1-p. \end{cases} \tag{3.3}$$
The synthetic datasets are generated by varying each parameter from default setting. We assign the default setting for parameters as follows, $N = 5000$, $m = 3$, $T = 10000$, and p (in Eq.3.3)$= 0.05$.
2. **Stock dataset (STK):** This stock dataset aggregated from Yahoo! Finance [1] consists of the daily information of all stocks in NYSE between 2004 and 2013. For

[1] Yahoo! Finance: http://finance.yahoo.com/

the scenario that we want to form a defensive investment portfolio, stocks that have a lower beta (not fluctuate with the market) and a trend of increasing in price than other stocks for a long period of time in the market are preferable. Therefore, we extract only 2 attributes including 200-day beta (β) and a 200-day slope of a regression line of close price (2 decimal precision). The system analyzes daily which stock acts or holds this characteristic for a long period of time and no better other choices in the market (skyline). Therefore, we take a daily change of these attributes as a snapshot. After data cleansing, this dataset contains 1630 stocks, 2000 snapshots and 1531069 update tuples (averagely 47% of entire monitored stocks).

3. **NBA dataset (NBA):** This NBA dataset aggregated by the authors in [70] consists of all historical NBA information on both game plays and player statistics between 1991 and 2004. We aim to find the skyline of players being active over that time period. By taking the end of each game as one snapshot, we need to compute the skyline after every game play. In summary, there are 1225 players to be monitored, 16423 matches played (snapshots) and 312086 update tuples in total. We selected 5 useful attributes from a record of each player in each match including play time in minutes (MIN), points made (PTS), total rebounds (TOT), field goal made (FGM) and field goal attempts (FGA). However, we extracted 3 attributes to evaluate players including $PTSA = \sum PTS / \sum MIN$, $TOTA = \sum TOT / \sum MIN$ and $FGR = \sum FGM / \sum FGA$. In this dataset, we can see that there are only at most 24 players changed (about 2% of entire monitored players) their statistics in a consecutive snapshot.

3.5.2 Comparison methods

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

1. **Naive method (baseline):** Compute the skyline by the default skyline algorithm using entire data objects in every snapshot ($D^0, D^1, \ldots, D^T$).

2. **Grid-based method (Grid-n):** We implemented the technique in [75] to prune by dividing the m-dimensional data space into n^m cells. Each cell is large $(c_w)^m$ units. In our experiments, we define $c_w = \max p_i^t[k]/n; \forall t \in \{0, 1, \ldots, T\}, \forall i \in$

$\{1, 2, \ldots, N\}$ and $\forall k \in \{1, 2, \ldots, m\}$. We varied n from 10, 30 and 50 respectively in each experiment. Then the candidates only in the grid cells that are not dominated by other cells are computed for skyline calculation while the rest can be safely pruned. This method is obviously not scalable because the number of grid cells can be tremendous in higher dimensionality. Also, this method requires the knowledge of data distribution, types of data attributes, data space to define proper values of grid cells' granularity and the space size. As in our experiments on real datasets, we had to find the maximum possible value in each attribute in advance to normalize the attribute values in order to fit in the specified data space. However, doing like this is not applicable in some real-life and real time applications, because this can be difficult and impossible for open-bounded value attributes. On the contrary, our proposed method has no restriction about the data space.

3. **LookOut [49]:** The continuous skyline computation method for single data insertion and deletion.

3.5.3 Measurements

We evaluated the performance by measuring the wall clock time of total execution time as well as counting the total number of processed candidates in each method. Normally, the skyline computation time is directly proportional to the number of processed candidates. In the naive method, there is only computation time due to skyline computation (SKY), but the grid-based method and our proposed method include both skyline computation time (SKY) and maintenance time (PRE and POST).

3.5.4 Results of the synthetic datasets

Parameter Tuning: The false positive tolerance threshold (θ_f) is only one system parameter in the proposed method. Here, as a preliminary experiment, we studied an effect of this parameter to decide a suitable value. At low θ_f, it incurred frequent MBR reconstructions while the size of candidates is reduced because of less false positives in S_M^t, and vice versa. It shows that setting low θ_f around 10–100 gives more preferable outcome than higher θ_f (1000–10000). In other words, paying some maintenance cost to

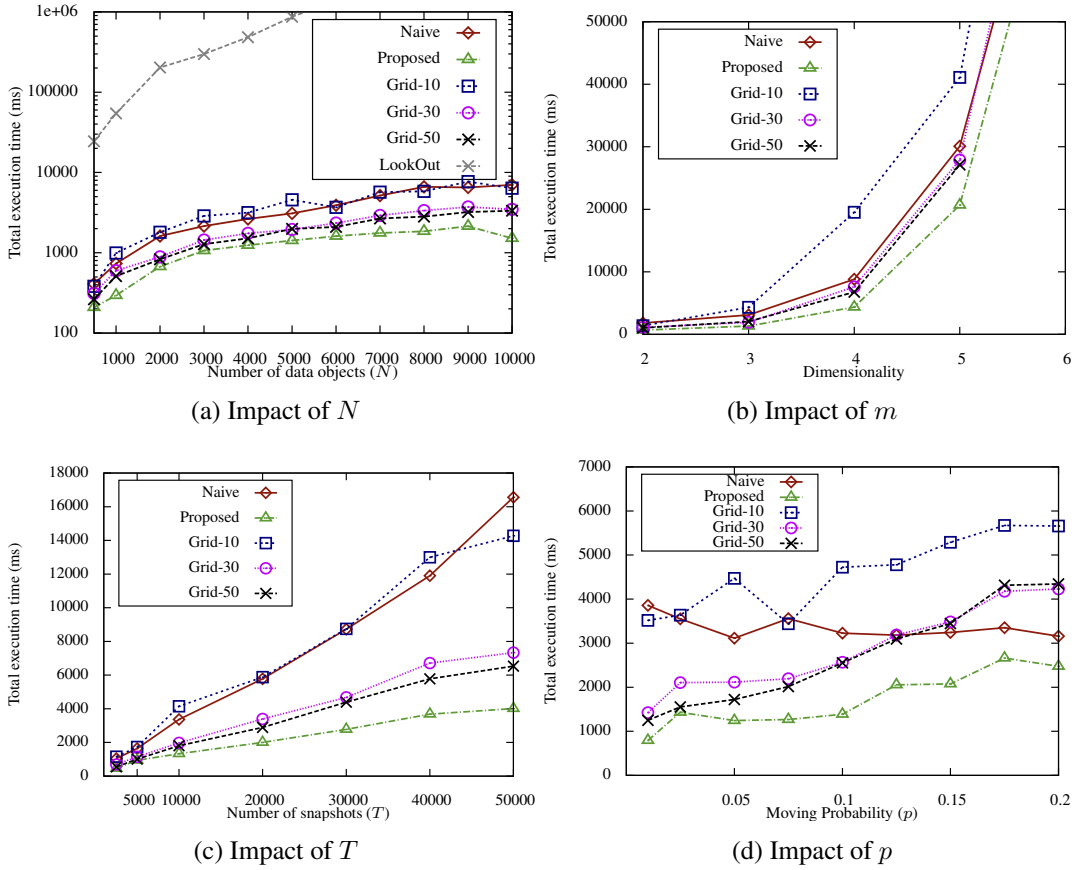


Figure 3.4: Results of the synthetic dataset

renew non-potential MBRs in S_M^t is worthy and able to reduce the overall computation time. Therefore, we assign θ_f equal to 50 as the default parameter in all experiments.

Impact of N: Increasing the number of objects, N, to be monitored directly affects the skyline computation time because of more candidates to be processed in each snapshot. As shown in Fig.3.4a, our proposed method runs faster than the others. As expected, the LookOut method performed worst than the others by a number of magnitudes, so we left out this method for the rest of the results. Even though Grid-10 can prune some candidates, their maintenance time is high due to the large number of blocks to be maintained. Its result turned poorer than the naive method. Regardless of the setting of the grid-based method, our proposed method processes less candidates (results omitted) compared with other methods and still outperforms the other methods in the

total computation time. This result ensures that our proposed method is scalable on a large number of data objects.

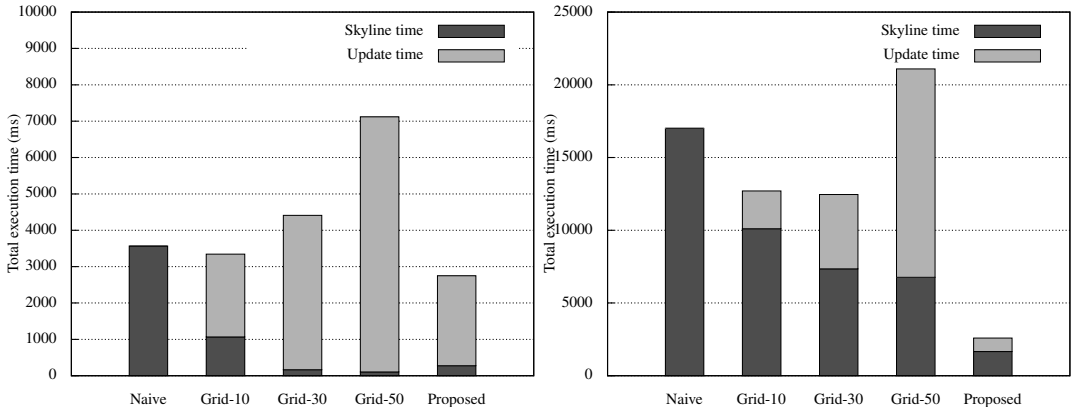
Impact of m: Normally, the cardinality of a skyline set is increasing exponentially with the number of dimensions, m, resulting in slower skyline computation. In Fig.3.4b, our proposed method outperforms the grid-based method because increasing the dimensionality also increases exponentially the number of maintained cells in the grid-based method, *i.e.*, n^m. We also tried experimenting on higher dimensionality than 5, but the grid-based method faces the errors due to the overflow of grid index number. At $m = 7$ (not shown in the figure), our proposed method still saves total computation time compared to the naive method by 20.5%.

Impact of T: In this setting, we simulated the result of all methods when using for long periods of time by increasing the number of snapshots, T. Normally, the total computation time grows linearly with this factor. However, in practice, it also depends on the cardinality of output affected by data distribution and data updates. The results show that our proposed method is more efficient than the others in long-term usage denoted by more gradual slope in Fig.3.4c.

Impact of p: In this setting, we study the effect when many data objects change their attribute values in each snapshot by increasing the probability of data object moving in a consecutive snapshot, p. According to the random walk model, increasing this probability scatters and deviates more data further from the initial points and produces different skyline output in each setup. However, this puts more work on the grid-based method to verify the correctness of membership in each cell as well as their cell dominance relationships and on our proposed method to verify the changes of MBRs in shape and dominance relationships. This maintenance cost worsens the grid-based method, and they become worse than the naive method as p increasing.

3.5.5 Results of the real datasets

We examine our proposed method's performance on 2 different real-life datasets, NBA and STK, which have totally different characteristics. NBA is quite larger than STK



(a) STOCK dataset (STK) (b) NBA dataset (NBA)

Figure 3.5: Results of the real datasets

in terms of the number of snapshots and the number of dimensions, but the moving probability (update ratio in each snapshot) is quite low averagely at only 2% for each snapshot while STK is at 47.6%. In this subsection, the results are reported in more details about the skyline computation time and the maintenance time.

In STK dataset, the data objects are frequently changed (many update tuples at a day tick) by its nature, and the moving probability is quite high averagely at 47%. Therefore, very high cost of maintenance for the grid-based method and our proposed method should be anticipated. As expected, the result of this dataset in Fig.3.5a reports the huge cost of maintenance time (Update time) in our proposed method and especially in the grid-based method. However, it is obvious that the more grid partitions, the higher cost of maintenance. Grid-30 and Grid-50 incur high cost of maintenance (Update time shown in Fig.3.5a), so the total execution time for these 2 methods eventually cannot beat the naive method. Nevertheless, our proposed method still saves the computation cost by 20% compared with the naive method.

Unlike STK, the NBA dataset consists of 3 attributes and more snapshots with lower average update rate. In Fig.3.5b, our proposed method achieves the best result among other comparison methods in both skyline calculation time (Skyline time) and maintenance time (Update time). Using the grid-based method can help pruning a lot in the STK dataset, but the pruning capability in this dataset is quite poor due to higher dimensionality and data distribution. In spite of lower number of candidates to be computed,

the cost of maintenance of Grid-50 makes it underperformed even the naive method.

3.6 Conclusions

In this chapter, we proposed an efficient method for skyline calculation when there are many data updates at each snapshot (timestamp). This is useful for analyzing historical (time-series) data archive as well as skyline computation on data update streams. In the assumed historical data series, the changes of data between consecutive timestamps are expressed and kept as update tuples. In practice, data insertion, deletion or any modification of a single data object between timestamp can totally change the final skyline set. Therefore, the naive method for this problem is to re-compute the new skyline set every timestamp (snapshot). This can be very expensive and time-consuming.

Our proposed method makes use of bounding boxes, *i.e.*, MBRs to summarize and represent a series of data snapshots of each data object. Due to the properties of MBRs and our technique to manage them, we can identify a smaller set of candidates for skyline computation by pruning non-potential data objects while the accuracy can be guaranteed. Moreover, we also discuss about the maintenance of our technique which is adaptive to the data changes in terms of temporal updates and data distribution.

We compared the performance of our proposed method through the experiments by using both synthetic dataset and 2 real datasets. The results obviously showed the benefits of our proposed method over the other methods by measuring the total execution time.

Reducing the response time in skyline monitoring through the algorithmic improvement in our proposed method has an impact on increasing user experience. Otherwise, our propose method is more scalable in terms of the number of data objects to be observed.

Chapter 4

Cost-minimizing methods for top-k and skyline monitoring

4.1 Introduction

A monitoring query is one of primitive tasks in sensor networks (SNs) for various applications, *e.g.*, environmental monitoring, disaster prevention and industrial monitoring. Consequently, monitoring query processing, *i.e.*, continuous query processing, has been extensively investigated in the research community, for example, skyline monitoring, top-k monitoring and kNN monitoring to name a few. Much research focuses on the problem of how to efficiently answer those diverse types of queries.

In terms of local computation, researchers aim to improve computation time and I/O to process acquired sensor data streams. In the aspect of SNs, they focus on reducing communication cost (data transfer) because sensors mostly have limited bandwidth and battery power, and communication cost directly affects battery lifetime. Therefore, the conventional techniques include dedicated in-network data aggregation and filtering. Note that, each proposed solution is application-specific, *i.e.*, the usage of SNs is limited to the agenda of the owners or organizations. In other words, the techniques are different based on the types of queries, so these methods are applicable for task-specific sensor networks and rarely used across types. As a result, the utilization of the sensors is not maximized.

With the development of cloud computing, a new paradigm computation for SNs

called *sensor cloud* aims to integrate multiple SNs and to deal with a massive volume and various fields of sensor data from forthcoming technologies of Internet of Things (IoT) [28], Smart Cities [56], etc. Such valuable sensor data can be commercialized as Sensing as a Service (S^2aaS).

S^2aaS [66] in sensor cloud abstracts all physical layers of sensor network integration. It provides users with the capability to effectively utilize provisioning virtual sensors, so the users can perform their desired sensing tasks by using various kinds of sensors in available wide-spread areas without technical difficulties. In commercial services [48], users pay the usage cost based on the amount of utilization (pay as you go) without the necessity of deploying their own infrastructure, while SNs providers and data contributors may receive rewards in the forms of rental fee or the cost of requested data.

With S^2aaS, a monitoring task can be processed with ease. This chapter firstly focuses on top-k monitoring query which delivers the best k-rank objects to a user periodically. This is substantial in view of the fact that a user would be interested in only a few best data objects (say k objects) at a time.

An application scenario is to monitor the k most vulnerable locations where a risk (*e.g.*, flood) is evaluated by precipitation, water level and wind speed in each region by using a sensor cloud service in (near) real-time. In addition, not only real-time applications but some analytic queries, for example, durable top-k query [88], also need to access historical time series data which can be provided by sensor cloud services. In such a scenario, values of data records in consecutive timestamps rarely jump but gradually change. Our work also employs this observation as an underlying assumption.

However, many of the conventional queries including top-k query and skyline query [10] are rank-aware queries where the calculation involves with comparisons between objects (aggregate query). This means all objects need to be retrieved and compared in order to give the correct answer in each timestamp even though in top-k query, for example, the cardinality of the answer set is only k that is much smaller than the total number of objects. Because we need to pay the cost of utilization based on the amount of requested data, this can cost a huge sum of expense.

In this chapter, we first tackle this problem and propose a cost-minimization monitoring framework for top-k queries where cost is referred to the expense for data access in sensor cloud services. To reduce the expense, our designed framework skips retrieving actual data tuples of unpromising objects to save the cost. Those skipped

data objects are modeled with the proposed uncertainty model, and data requests are issued based on the necessity on querying. We then develop efficient approximate top-k processing methods with a quality guarantee on those uncertain and actual data objects. Unlike the work in [1] which tries to answer approximate aggregate queries with bounded response times, our goal is to answer continuous top-k queries with bounded expense budgets.

Even though our proposed top-k processing method brings about more computational complexity, in this chapter, we also presents an implementation of our framework on well-known and de facto parallel-distributed frameworks for cloud computing, *i.e.* Hadoop MapReduce and Spark, as well as its enhanced scheme to further speed up calculation time and reduce computation loads on the cloud. The results from extensive experiments on two real datasets show that, by employing our framework, the expense can be cut at least by half while the accuracy gets a perfect score. Furthermore, the additional computation time compared with that of a traditional exact method can be suppressed by using parallel computing on the cloud.

We also shows that this similar framework can also be applied to the case of skyline monitoring without difficulties, and the proposed method yields the same benefits as same as that of top-k monitoring.

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

- We design a framework to reduce the expense for top-k monitoring in sensor cloud services.

- We developed efficient methods of approximate top-k query processing with a probabilistic guarantee on the designed framework.

- We thoroughly conduct extensive experiments to study the efficiencies in terms of expense, time and accuracy on two real-world datasets.

- We also show that the same underlying framework can be easily extended to be used for skyline monitoring.

4.2 Related Work

Top-k query processing has been actively researched for many years. At first, many techniques in top-k query processing on centralized databases have been proposed [36]. Later, there have been a number of works on top-k query processing in different contexts such as continuous top-k [91], distributed top-k [87], top-k dominating [95] and durable top-k [88].

All of the works mentioned above can be roughly classified as either centralized or distributed systems. The works for centralized systems concentrate on indexing, caching and pruning methods to reduce computations resulting in more efficient top-k processing in terms of number of I/O and execution time. For distributed systems, *e.g.*, wireless sensor networks (WSNs) [91] and peer-to-peer systems (P2P) [87], the objectives are to reduce the response time, the number of contacted nodes and the traffic incurred by in-network top-k processing. In sensor cloud services, these specific issues no longer exist because data can be accessed directly from the cloud. To the extent of our knowledge, we are the first to address the problem of cost minimization of monitoring queries in sensor cloud services.

Data collection can be inaccurate due to reading errors, inaccurate sensors, etc. Due to the existence of data uncertainty, a new class of query called probabilistic query has been actively studied [2]. The definition of uncertain data can be roughly divided into two categories: (1) tuple-level uncertainty (sometimes called existential probability) – the probability of presence or absence of a data tuple found in [32, 59, 69] and (2) attribute-level uncertainty – uncertainty of an attribute represented by probability mass functions, parametric statistical models, etc. [54, 68, 98].

Our assumed uncertainty model is classified in the attribute-level uncertainty. However, the source of uncertainty in our work is different from the works above that assume the uncertainty happens to data at the source of information while our work assumes the collected data are accurate but the uncertainty of data occurs because of the oldness (age of data). Our data model is similar to moving objects, *i.e.*, spatio-temporal data. There are various models to represent uncertain moving objects, for example, free moving region [15], diamond [24] and bivariate normal distribution [57].

4.3 Preliminaries

4.3.1 Sensor Cloud

Sensor cloud is a model extended from cloud computing to manage and integrate physical sensors and turn them into a number of services. Likewise cloud computing, sensor cloud is scalable in terms of storage, computing capability, the number of users and services. Sensor cloud, unlike general cloud computing, not only providing on-demand computing and storage resources but also convenient access to sensor data on a pay-as-you-go basis. Definitely, the expense of using cloud services can be divided into many categories, *e.g.*, the expenses of computing cores, storage, backups and databases. In this paper, we focus on the expense which is additionally included apart from the base costs of computing and storage in sensor cloud, *i.e.*, the expense of sensor data access. This expense can be very high because sensor data are generated with different costs, *e.g.*, types of sensors, maintenance cost, deployment difficulty and data acquisition. Therefore, in this paper, the word *expense* refers to the cost of data access incurred in sensor cloud services regardless of other expenses.

4.3.2 Data Model

Sensor cloud integrates multiple-attribute data, *i.e.*, a single data object represented by a tuple of values, from many diverse physical sensors, and then performs data preprocessing on those data before storing into the storage and opening on-demand or real-time access to users/applications. Normally, for power-efficient data aggregations in sensor networks, sensors perform data sensing in duty-cycle, *i.e.*, sensors periodically sense values when they are active and put themselves to sleep, so the availability of data in sensor cloud services is not continuous but snapshot-like that each data record is associated with a timestamp. In addition, sensor data provided in sensor cloud services can be from other sources beyond physical sensors, *i.e.*, virtual sensors where data collected from social networks and web data. For a specific application, we are usually interested in the same set of attributes from a number of sensors rather than considering them as heterogeneous sensors. Hence, we assume that sensors supply values of the same set of attributes in each timestamp.

Given a set of m attributes including $A_T = \{A_1, A_2, \ldots, A_m\}$, at each timestamp

t, a physical/virtual sensor M_i dynamically generates a data record d_i^t where i is the unique identifier (id) of the sensor i, and $d_i^t = (d_i^t[0], d_i^t[1], \ldots, d_i^t[m]) \in \mathbb{R}^m$ is a data tuple where each element respectively represents the values of each attribute in A_T. Taking only the first attribute in a sensor into account, a sequence of $\{d_i^0[0], d_i^1[0], \ldots\}$ is time series. We define $D^t = \bigcup_{i=1}^{N_D} d_i^t$ as a set of actual data tuples of all N_D sensors at timestamp t. It is noted that the actual values of data tuples from one sensor vary with t. Nonetheless, in the two consecutive snapshot t and $t+1$, the values of the data tuples can be changed or unchanged.

4.3.3 Data Access

In a sensor cloud service, we assume that the cost of data access is charged per tuple to the users/applications. A user requests a data tuple via API by a key which is a pair of (i, t) where i indicates the sensor identifier and t indicates the timestamp of the data tuple the user wants to acquire. Based on the applications, t is decided by users (pull-based access), *e.g.* event monitoring (requesting latest t) or historical data analysis (requesting old timestamp t). We assume that users know the data availability at time t, *e.g.*, data update on the hour. Then the sensor cloud service will reply back with an augmented tuple (i, t, d_i^t) where d_i^t is an m-dimensional tuple of attribute values. This model is commonly found in key-value databases, *e.g.*, HBase[1], MongoDB[2], etc., and commercial data vendors, *e.g.*, Microsoft Azure Marketplace[3] (Fig.4.1).

4.3.4 Top-k monitoring query model

A top-k query $q = (f, k)$ from a user is defined by a linear scoring function f in which its weightings can be represented as a vector $\mathbf{w} = (w_1, w_2, \ldots, w_m)$ where w_j stands for positive weighting at j-th dimension and a parameter k. We focus on linear combination functions such that a score of a data record d_i is $f(d_i^t) = \sum_{j=1}^{j=m} w_j d_i^t[j]$. This class of functions is common because it represents how a user gives priority (weighting) to each factor. A set of weightings can be acquired by many possible ways, for example, given directly by users and adapted from the studies of each objective. A scoring function is

[1] http://hbase.apache.org/

[2] https://www.mongodb.org/

[3] https://datamarket.azure.com/browse/data

Figure 4.1: Microsoft Azure Marketplace providing data access service charged by the number of requests

monotonic, that is if $d_a[i] \leq d_b[i]$ for all $1 \leq i \leq m$, then $f(d_a) \leq f(d_b)$. The value k defines the number of desired data records.

The final top-k answers of query q at timestamp t are denoted by $T_k^t = topk(D^t) = \{a_1^t, a_2^t, ..., a_k^t\}$ where $a_i^t \leq a_j^t$, $i > j$ and $T_k^t \subseteq D^t$. By the definition of final top-k answers, $\forall d_x^t, d_y^t : d_x^t \in T_k^t, d_y^t \in D^t \backslash T_k^t \rightarrow f(d_x^t) \leq f(d_y^t)$. It is noted that we prefer to objects whose scores are lower. Top-k monitoring aims to identify $T_k^0, T_k^1, \ldots, T_k^{N_t}$ where N_t is the number of snapshots.

4.4 Cost Minimizing Framework

Top-k query is a rank-aware query. Even though the cardinality of the final top-k answers is exactly equal to k, but, to calculate the correct answers, D^t which has a size of N_D must be retrieved in each timestamp t. This means that $N_D - k$ retrieved objects are wastefully collected but never used for top-k monitoring purposes. Remind that data access is charged per tuple, this costs a huge amount of money.

4.4.1 Problem Statement

In this paper, we investigate the problem of reducing the expense in sensor cloud services for top-k monitoring queries. The objectives are to keep the accuracy of the final top-k results high while the expense must be significantly reduced compared with the baseline approach.

4.4.2 Observations

A sensor generates a sequence of data points (time series). In many contexts, *e.g.*, finance and meteorology, time series analysis is used for representing trends and prediction. Therefore, we may be able to predict with high confidence based on the historical records that d_i^t is very unlikely to be included in the top-k results T_k^t because d_i^{t-1} is too far from becoming top-k answers compared with other candidates. From this observation, we may skip fetching d_i^t to save cost, while the last known value of this object is d_i^{t-1}. Definitely, if we keep deciding not to fetch this object in the next snapshot, *i.e.*, d_i^{t+1}, $d_i^{t+2}, \ldots$, we gradually lose the confidence that d_i^{t+1}, $d_i^{t+2}, \ldots$ are unlikely to be included in $T_k^{t+1}, T_k^{t+2}, ...$ because the values may have been changed significantly from the latest-retrieved value. Eventually, $d_i^{t'}$ must be retrieved to ensure the confidence of the status of $d_i^{t'}$ in $T_k^{t'}$ at some $t' \geq t$. For example, let $t' = t + 5$, this means we can save cost not to access 4 times.

4.4.3 Uncertain data model

In this section, we illustrate how to model the uncertainty of data tuples with respect to their oldness. The oldness is defined as the difference of timestamps between latest-retrieved timestamp and the current timestamp.

Considering only a single attribute, we model the change in values of the attribute in 2 consecutive snapshots following a normal distribution $\mathcal{N}(\mu, \sigma^2)$ where μ is the mean and σ^2 is the variance as shown in Eq.4.1.

$$d_i^{t+1}[j] \sim \mathcal{N}(d_i^t[j], \sigma[j]^2) = d_i^t[j] + \mathcal{N}(0, \sigma[j]^2) \tag{4.1}$$

This time series form a Gaussian random walk which is a random walk model having a step size that is based on Gaussian distribution. Random walk model is used or

Table 4.1: Example of D^5 and U of 8 objects at $t = 5$

D^5

i	d_i^5
1	$(18.42, 6.21)$
2	$(10.77, 30.03)$
3	$(20.14, 55.45)$
4	$(48.12, 13.15)$
5	$(39.05, 42.14)$
6	$(18.82, 91.11)$
7	$(72.25, 29.86)$
8	$(82.00, 72.66)$

U and $t = 5$

i	t'	$d_i^{t'}$	Δt
1	5	$(18.42, 6.21)$	0
2	5	$(10.77, 30.03)$	0
3	5	$(20.14, 55.45)$	0
4	4	$(46.78, 10.58)$	1
5	3	$(41.25, 48.98)$	2
6	2	$(28.65, 86.01)$	3
7	2	$(86.09, 26.60)$	3
8	2	$(82.56, 72.64)$	3

underlying in real world time series data, *e.g.*, financial data [35].

From Eq.4.1, let n be the number of steps from t then

$$d_i^{t+n}[j] \sim d_i^t[j] + \mathcal{N}(0, n\sigma[j]^2) \tag{4.2}$$

Therefore, considering m independent attributes

$$d_i^{t+n} \sim d_i^t + \mathcal{N}(0, n\Sigma) \tag{4.3}$$

where $\Sigma = \begin{bmatrix} \sigma[1]^2 & & \\ & \ddots & \\ & & \sigma[m]^2 \end{bmatrix}$ is a covariance matrix and $\sigma[i]^2$ is the variance of attribute i.

We use this explained model to define uncertainty of the data objects at timestamp t when the latest-retrieved timestamp is t', so the oldness is defined as $\Delta t = t - t'$ where $t \geq t'$. To distinguish between the actual values at the present timestamp and the values with uncertainty, we define a new table called U. The cardinality of U is N_D as same as D^t. The data objects in U includes $\{u_1, u_2, \ldots, u_{N_D}\}$ which each element u_i is a tuple of $(t', d_i^{t'})$ indicating that the object d_i was latest-retrieved at timestamp t' having values $d_i^{t'}$.

Table 4.1 depicts an example of D^5 (the list of actual data at $t = 5$) and U. They contain 8 objects including $i = \{1, 2, \ldots, 8\}$. Each record has different degrees of uncertainty. In the table, u_1, u_2 and u_3 have been retrieved in this timestamp, so their

values are exactly the same as d_1^5, d_2^5 and d_3^5. On the other hand, u_6, u_7 and u_8, for instance, have not been retrieved since timestamp 2 ($\Delta t = 3$).

Variances from datasets

Because the uncertain data model is designed based on the assumption that values change following the Gaussian model, an important component for this model is the covariance matrix Σ, *i.e.*, the variances of each independent attribute. In other words, these define the expected step size in each change of attribute data. In some cases, we can roughly predict or approximate the acceptable variance for an attribute. For example, temperature generally does not change more than 5 degrees (positive and negative) in the next timestamp (*e.g.*, within 10 mins). We may presume that 95% the step size is between [0,5] (ranging between [-5,5]) that means $2\sigma = 5$. Therefore, we may set $\sigma^2 = 25/4 = 6.25$ as a variance of this attribute.

However, for unfamiliar or unknown attributes, the variance can be approximated from the existing historical records using Eq.4.4.

$$\sigma^2[j] = Var(\{\Delta d | \Delta d = |d_i^t[j] - d_i^{t-1}[j]|\}) \tag{4.4}$$

where $j \in \{1, 2, \ldots, m\}$, $i \in \{1, 2, \ldots, N_D\}$ and $t > 0$.

In addition, for more or less precision, we attached the multiplier factor τ where $\tau > 0$ to reflect more/less safety margin from the estimated variances resulting in Eq.4.5.

$$\Sigma = diag(\tau\sigma^2[1], \tau\sigma^2[2], \ldots, \tau\sigma^2[m]) \tag{4.5}$$

4.4.4 Epsilon top-k query processing (ε-top-k)

Unlike traditional query processing which works on a complete set of actual data values (D^t), we suppose that, in each timestamp, some data tuples are not retrieved in order to save cost, so some data tuples are not up-to-date and contain uncertainty due to the oldness as explained in Section 4.4.3. Therefore, in this section, we propose an approximate top-k query processing method working on such a situation. Nonetheless, the approximate top-k answers must satisfy the statistical error bound.

Definition 4.4.1. An ε-top-k query returns the up-to-date answer set $T_{\varepsilon,k}^t$ that, based on a given uncertain model, with the probability higher than ε that $T_{\varepsilon,k}^t$ calculated from U is not different from T_k^t calculated from D^t.

In top-k query processing, top-k answers (T_k^t) and non-top-k objects ($D^t \backslash T_k^t$) can be divided by a hyperplane $H : (w_1)x_1 + (w_2)x_2 + \ldots + (w_m)x_m = \theta$ where $\theta = f(a_k^t)$. We define the subspace $S : (w_1)x_1 + (w_2)x_2 + \ldots + (w_m)x_m \leq \theta$ as the top-k subspace because only and all top-k answers must fall in this linear subspace. Considering only an uncertain object $u_i \in U$ and a ranked list of k certain objects I_k^t (alike T_k^t), the probability that u_i is included in I_k^t, *i.e.*, the actual values of u_i will replace some intermediate answers in I_k^t, is equal to the following p_{u_i}.

$$p_{u_i} = \frac{\int_S d_i^{t'} + \mathcal{N}(0, (t-t')\Sigma)ds}{\int_{\mathbb{R}^m} \mathcal{N}(0, (t-t')\Sigma)ds} \tag{4.6}$$

Therefore, the probability that I_k^t is already the actual top-k answers is equal to $1 - p_{u_i}$.

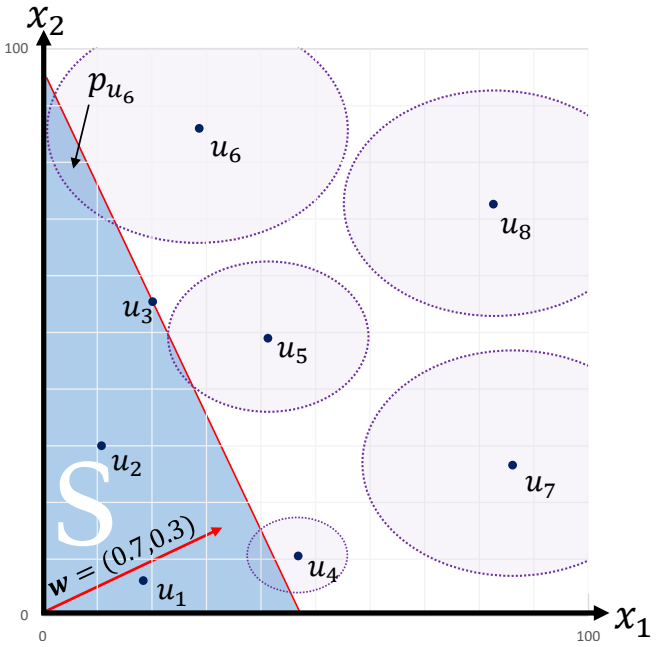
This time, considering a ranked list of k certain objects I_k^t and the entire table U except I_k^t, the probability that I_k^t is already the actual top-k answers is as follows:

$$t_p = \prod_{u_i \in U \backslash I_k^t} (1 - p_{u_i}) = \prod_{u_i \in U \backslash I_k^t} (1 - \frac{\int_S d_i^{t'} + \mathcal{N}(0, (t-t')\Sigma)ds}{\int_{\mathbb{R}^m} \mathcal{N}(0, (t-t')\Sigma)ds}). \tag{4.7}$$

I_k^t is acceptable as the approximate top-k answers w.r.t. ε ($T_{\varepsilon,k}^t$) if and only if $t_p \geq \varepsilon$.

Running example: Fig.4.2 illustrates an example of uncertain objects from Table 4.1 ($t = 5$) with uncertainties denoted by circles around those points. At $t = 5$, we know that u_1, u_2 and u_3 have been already updated as $\Delta t = 0$. With $\mathbf{w} = (0.7, 0.3), k = 3$, $I_3^5 = \{u_1, u_2, u_3\}$, so $\theta = f(u_3) = 20.14 \times 0.7 + 55.45 \times 0.3 = 30.733$. A tuple $(\mathbf{w}, \theta)$ describes the subspace S as shown in the figure. The upper table on the right shows the example calculation of p_{u_i} for each object. It is noted that p_{u_i} where $u_i \in I_3^5$, which is supposed to be 0, does not need to be calculated, so n/a is labeled instead. Therefore, t_p in Eq.4.6 is equal to 0.7497. This means this I_3^5 is acceptable as εT_3^5 if $\varepsilon \leq 0.7497$. Otherwise, we need to an execution plan to update U to increase t_p.

Strategies to increase t_p to meet the requirement: The parameter ε controls the confidence of approximate answers. It is obvious that the probability t_p turns small because of uncertainty of existing data in $U \backslash I_k^t$. Because uncertainties of some uncertain data objects in $U \backslash I_k^t$ can be eliminated by requesting actual values at the current timestamp



p_{u_1}	n/a
p_{u_2}	n/a
p_{u_3}	n/a
p_{u_4}	0.15
p_{u_5}	0.02
p_{u_6}	0.10
p_{u_7}	0.00
p_{u_8}	0.00

$I_3^5 = \{u_1, u_2, u_3\}$

$\theta = f(u_3) = 30.733$

$$t_p = \prod_{u \in U \setminus I_3^5} (1 - p_u) = 0.7497$$

Figure 4.2: Illustration of uncertain objects in Table 4.1

(actual data), t_p can be increased and meet the required ε. There are three possible cases when including up-to-date values to U: (1) t_p changed due to the decrease of uncertainty (2) t_p changed due to the changes of answers in I_k^t resulting in the change of θ and consequently the change of integral domain S (3) both cases above.

4.4.5 Multidimensional Integration

Calculating t_p in Eq.4.7 involves numerical integration. This is not limited to only 2-dimensional data as in Fig.4.2, but also an integral of the multivariate Gaussian distribution on subspace domains. These have complex closed form expressions.

Monte Carlo integration (also used in [59]), which is a sampling-based method, is commonly employed in such tasks. Monte Carlo integration draws random points over some domain A which is a superset of A'. Then the area (volume, or m-dimensional content, etc.) of A' is estimated by the area of A multiplied by the fraction of points falling within A'.

In the same way, we estimate p_{u_i} by drawing N_S random samples (the number of samplings) from a normal distribution $\mathcal{N}(d_i^{t'}, (t - t')\Sigma)$. We estimate the probability

$p_{u_i} \approx N'_S/N_S$ where N'_S is the number of random samples that fall in the subspace S. This approach can be applied to other uncertainty models as long as random samples can be drawn from the models.

4.5 Proposed Algorithms

4.5.1 Multiple-round evaluation

In this section, we propose a method to calculate $T^t_{\varepsilon,k}$. The main idea is based on a greedy approach that we try to remove the uncertainty of the data object that affects t_p most first. Then, the same procedure is re-evaluated until the probability t_p meets the requirement ε.

Algorithm 7 shows the procedure of our proposed framework. At the first timestamp $t = 0$, U is constructed by requesting all data objects (line 1). This costs exactly N_D data accesses, but it is done once in the initialization. Because all data objects are certain, T^0_k which is the actual top-k answers can be calculated and returned as $T^0_{\varepsilon,k}$ (line 2). We also initialize intermediate results I as T^0_k (line 3).

For each latter timestamp t when $t > 0$, data objects in U which appear to be the top-k answers of the previous timestamp, *i.e.*, I are decided to be first requested and updated on U (lines 4–7) because we suppose that the top-k answers of the previous timestamp are potential candidates to be included in the top-k answers in the current timestamp. After that, in lines 8–23, intermediate top-k answers denoted by I are firstly calculated using the recently-updated U. This may also change θ and S. Then the probability t_p according to Eq.4.7 is estimated. Meanwhile, the index of the uncertain object, which affects t_p the most (uncertain object that gives $\min_{u_i \in U \setminus I}(1 - p_{u_i})$) is kept denoted by i_m (lines 17–19).

At lines 20–22, t_p is verified whether it passes the given condition ε. If not, the data object i_m in U is requested and updated on U. Then, the procedure from line 9 is repeated to calculate the new intermediate results I and re-evaluate t_p until $t_p \geq \varepsilon$ (See Lemma 8). Finally, I is returned as the final approximate top-k answers at timestamp t ($T^t_{\varepsilon,k}$).

Lemma 8. *In the loop between lines 8–23, as the algorithm requests the data object i_m (lines 20–22), t_p increases in every iteration.*

Proof. Considering I consisting of k objects with the last-ranked element a_k, if $d^t_{i_m}$ has been retrieved and updated on U, the next iteration of I, say I', is calculated by using the updated U. This results in two possible cases as follows: (1) The k objects of I' are the same set as I – this means that the last-ranked object is still a_k, and θ is still unchanged, but $(1 - p_{i_m})$ which is used to be $\min_{u_i \in U}(1 - p_{u_i})$ becomes 1. Therefore, $t_p = \prod_{u_i \in U \setminus I'}(1 - p_{u_i})$ must be increased. (2) The k objects of I' are different from I – this means $d^t_{i_m} \in I'$, so let the last-ranked object be a'_k (a_k was replaced), we conclude that $f(a'_k) \leq f(a_k)$. As a result, the subspace $S' : (w_1)x_1 + (w_2)x_2 + \ldots + (w_m)x_m \leq \theta'$ where $\theta' = f(a'_k)$ becomes a subspace of S. From Eq.4.6, $\forall u_i \in U : p_{u_i}$ will be decreased and also $(1 - p_{i_m}) = 1$ resulting in higher $t_p = \prod_{u_i \in U \setminus I'}(1 - p_{u_i})$. □

4.5.2 Single-round evaluation

An impractical drawback of Algorithm 7 is that delivering $T^t_{\varepsilon,k}$ requires multiple times of p_{u_i} calculation for all u_i in $U \setminus I$ (lines 14–19). The worst case is up to $O(|U|)$ times per timestamp, and this calculation involves with the multi-dimensional integration explained in Section 4.4.5 which is very computing-intensive especially in a high number of samplings (N_S) for accurate estimations.

To solve this problem, we develop another efficient algorithm which requires only one time of p_{u_i} calculation for all $u_i \in U$. In fact, the access cost of the following algorithm is higher than Algorithm 7, but it is more practical for computing. The main idea of this algorithm is that we try to find intermediate k certain objects stored in I first by iteratively requesting data objects until I is fixed. Then, we calculate p_{u_i} one-time for all $u_i \in U$ w.r.t I and make a table of U sorted by $1 - p_{u_i}$. After that, we request the data objects by using that sorted table to update U, so that $topk(U)$ is guaranteed to be $T^t_{\varepsilon,k}$.

The pseudo-code is shown in Algorithm 8. Lines 1–7 shares the same idea with Algorithm 7. Lines 8–16 try to find intermediate k certain objects stored in I as well as update U with latest data values. It ensures that, given $I = topk(U)$, $\forall u_i \in I : u_i.t' = t$ (values at this timestamp). Then, p_{u_i} is calculated and stored in a max heap $(1 - p_{u_i}, u_i)$ for $u_i \in U$ by using $(1 - p_{u_i})$ as keys (lines 17–20). In lines 21–28, t_p is initialized as 1.0

Algorithm 7: Proposed Framework: Multiple Round

Input: N_t, f, k, ε

Output: $T^0_{\varepsilon,k}, T^1_{\varepsilon,k}, \ldots, T^{N_t}_{\varepsilon,k}$

$U \leftarrow$ retrieve all data object at timestamp 0 (D^0)
Calculate T^0_k using U and return it as $T^0_{\varepsilon,k}$
$I \leftarrow T^0_k$
for *timestamp* $t = 1 \rightarrow N_t$ **do**
    for *object* $o_i \in I$ **do**
        $d^t_i \leftarrow$ request data object i at timestamp t
        Replace $u_i \in U$ with (t, d^t_i)
    do
        $I \leftarrow topk(U)$
        $\theta \leftarrow f(a^t_k)$ where a^t_k is the k-rank object in I
        $t_p \leftarrow 1.0$
        $p_m \leftarrow 1.0$
        $i_m \leftarrow null$
        for *object* $u_i \in U \backslash I$ **do**
            Calculate p_{u_i} (Eq.4.6)
            $t_p \leftarrow t_p(1 - p_{u_i})$
            if $(1 - p_{u_i}) \leq p_m$ **then**
                $p_m \leftarrow (1 - p_{u_i})$
                $i_m \leftarrow i$
        if $t_p < \varepsilon$ **then**
            $d^t_{i_m} \leftarrow$ request object i_m at timestamp t
            Replace $u_{i_m} \in U$ with $d^t_{i_m}$
    while $t_p \geq \varepsilon$
    Return I as $T^t_{\varepsilon,k}$

and the top element (the highest $1-p_{u_i}$ in the heap) is peeked (examined) and multiplied by t_p. If that result is still higher than ε, that top element will be polled (removed), and t_p is set to that result. Otherwise, line 29–32 are continued with remaining elements in the heap. In lines 29–32, data objects remaining in the heap are requested and updated on U. Finally, I is re-calculated again using the updated U and returned as $T^t_{\varepsilon,k}$.

The following lemma proves that the final I calculated from U, *i.e.*, $T^t_{\varepsilon,k}$ has t_p that $t_p \geq \varepsilon$.

Lemma 9. *$T^t_{\varepsilon,k}$ from Algorithm 8 gives an approximate set of answers with $t_p \geq \varepsilon$.*

Proof. We show a proof by contradiction assuming that $t_p < \varepsilon$. From lines 23–28, let H_1 be a set of polled objects from line 25, the algorithm shows that $\prod_{u_i \in H_1}(1-p_{u_i}) \geq \varepsilon$. Between lines 29–32, let H_2 be a set of polled objects from line 30, these objects are decided to be requested, and U is updated. Doing this also removes the uncertainty of data objects in H_2, therefore, $\prod_{u_i \in H_2}(1-p_{u_i}) = 1$. It is noted that $U = H_1 \cup H_2$ and $H_1 \cap H_2 = \emptyset$. This can be divided into 2 cases: (1) All updated data objects in H_2 do not belong to $topk(U)$, this means the original I after line 16 is equal to $topk(U)$. Therefore, $t_p = \prod_{u_i \in H_1}(1-p_{u_i}) \prod_{u_i \in H_2}(1-p_{u_i}) \geq \varepsilon$. (2) Some updated data objects in H_2 belong to $topk(U)$, this means the last-ranked object of $topk(U)$ is different from the original I after line 16. Let a_k be the last ranked object of the original I and a'_k be the last ranked object of $topk(U)$. We conclude that $f(a'_k) \leq f(a_k)$. In the same way of the proof of Lemma 8, the subspace $S' : (w_1)x_1 + (w_2)x_2 + \ldots + (w_m)x_m \leq \theta'$ where $\theta' = f(a'_k)$ becomes a subspace of S. As a result, $\prod_{u_i \in H_1}(1-p_{u_i})$, which is greater or equal to ε, further increases while $\prod_{u_i \in H_2}(1-p_{u_i}) = 1$. Therefore, in both cases, $t_p = \prod_{u_i \in H_1}(1-p_{u_i}) \prod_{u_i \in H_2}(1-p_{u_i}) \geq \varepsilon$. This contradicts the assumption. □

4.6 Spark-based Implementation

According to Algorithm 8, lines 18–20, the calculation of p_{u_i} is computing-intensive. In a cloud platform, MapReduce is a common computing model, and that calculation can be computed in parallel as well as distributed.

Recently, Apache Spark[4], which has been developed to overcome traditional MapReduce's shortcomings that it is not optimized for iterative algorithms, has been often com-

[4]https://spark.apache.org/

Algorithm 8: Proposed Framework: Single Round

Input: N_t, f, k, ε

Output: $T^0_{\varepsilon,k}, T^1_{\varepsilon,k}, \ldots, T^{N_t}_{\varepsilon,k}$

$U \leftarrow$ retrieve all data objects at timestamp 0
Calculate T^0_k using U and return it as $T^0_{\varepsilon,k}$
$I \leftarrow T^0_k$
for *timestamp* $t = 1 \rightarrow N_t$ **do**
  for *object* $o_i \in I$ **do**
    $d^t_i \leftarrow$ request data object i at timestamp t
    Replace $u_i \in U$ with d^t_i
  do
    $I \leftarrow topk(U)$
    $F \leftarrow$ false
    for *object* $u_i \in I$ **do**
      if $u_i.t' - t > 0$ **then**
        $d^t_i \leftarrow$ request data object i at timestamp t
        Replace $u_i \in U$ with d^t_i
        $F \leftarrow$ true
  while $F =$ *true*
  $H \leftarrow$ an empty max-heap (p, u) using p as a key
  for *object* $u_i \in U$ **do**
    Calculate p_{u_i} (Eq.4.6)
    $H.insert((1 - p_{u_i}), u_i)$
  $t_p \leftarrow 1.0$
  $F \leftarrow$true
  while $F =$ *true* **do**
    if $(H.peek().p)t_p \geq \varepsilon$ **then**
      $(p, u_i) \leftarrow H.poll()$
      $t_p \leftarrow (p)t_p$
    else
      $F \leftarrow$false
  while H *is not empty* **do**
    $(p, u_i) \leftarrow H.poll()$
    $d^t_i \leftarrow$ request data object i at timestamp t
    Replace $u_i \in U$ with d^t_i
  $I \leftarrow topk(U)$
  Return I as $T^t_{\varepsilon,k}$

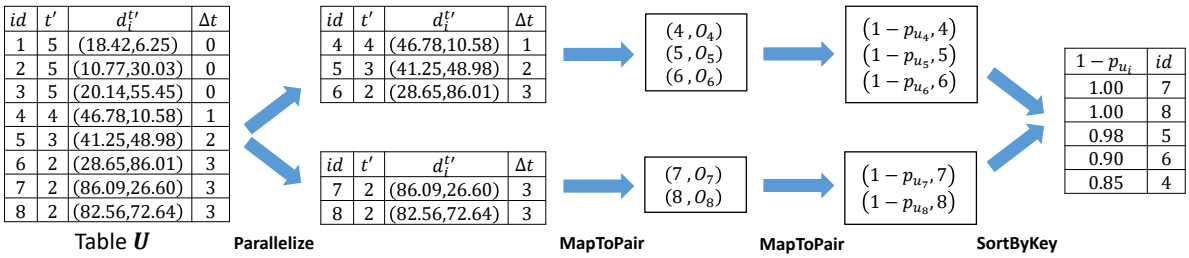


Figure 4.3: Workflow of Spark-based single round evaluation at timestamp 5

pared with MapReduce in terms of ease of use and performance. It turns that Spark is better than MapReduce according to lazy computation and use of main memory. Thus, in this paper, the implementation, operations are based on Apache Spark, even though both can be applied.

Fig.4.3 (the same values used in Fig.4.2) shows the workflow how to compute t_p in Spark which is equivalent to Algorithm 8 lines 17–32. In the first step, table U (only $\Delta t > 0$) is partitioned and parallelized across working nodes (distributed dataset creation). Then, in each parallelized table, each record is transformed to a pair of id i and its corresponding object (collections, *e.g.*, $d_i^{t'}, \Delta t, \Sigma, \mathbf{w}, \theta$ and N_S). After that, each pair is calculated for p_{u_i} and transformed to a pair of $(1 - p_{u_i})$ and its id i. Finally, the final output is collected by descending order of keys (SortByKey). Using this table (as if it is the max heap in line 17), we can easily identify which data object needs to be retrieved in lines 29–32. In Fig.4.3, given $\varepsilon = 0.9$, u_6 and u_4 need to be retrieved to ensure the requirement.

According to the experiment results described in details in Section 4.8, our proposed framework can reduce the expenses, *i.e.*, reducing the number of data accesses. Surprisingly, even though, our proposed framework incurs huge additional computation workloads, using multiple nodes to help computing intensive tasks can shorten the execution time, and, with enough computing resources, the total execution time is less than the conventional exact method because of less number of data accesses (lower total data access time).

4.7 Enhanced Approach

As mentioned, the calculation of p_{u_i} is a key time-consuming task. This task is repetitively executed though with different parameters in each iteration. It is noted that candidates which are still far from becoming the top-k answers will have $p_{u_i} = 0$. Since the size of k is supposed to be much smaller than N_D, a lot of pairs after p_{u_i} calculation get $p_{u_i} = 0$, *i.e.*, $(1 - p_{u_i} = 1.0)$. Those objects are not taken into consideration and wastefully utilize the computing resources. Especially in a cloud which is physically a computer cluster, this incurs communication between nodes resulting in long latency. The main concept of the following enhancement is to reduce the number of unnecessary calculations of some pairs (pruning) by using existing knowledge from the historical calculation, *i.e.*, views and indexes.

Before introducing the algorithms, the following definition and lemmas are the basis of the techniques.

Definition 4.7.1. (Dominance of points) [8, 54] The object d_1 dominates d_2 denoted by $d_1 \preceq d_2$ if and only if $d_1[j] \leq d_2[j]$ for all $j \in \{1, 2, \ldots, m\}$ and $t_1 \geq t_2$.

Lemma 10. *For any monotonic function f, if d_1 dominates d_2, then $f(d_1) < f(d_2)$ (Proof omitted).*

Definition 4.7.2. (Dominance of uncertain objects) An uncertain object u_1 dominates u_2 denoted by $u_1 \preceq u_2$ if and only if $u_1.d_1^{t_1}[j] \leq u_2.d_2^{t_2}[j]$ for all $j \in \{1, 2, \ldots, m\}$ and $t_1 \geq t_2$.

Lemma 11. *Given 2 pairs, (p_{u_1}, u_1) and (p_{u_2}, u_2) evaluated in the same subspace S which is described by $(\boldsymbol{w}, \theta)$, if $p_{u_1} = 0$ and $u_1 \preceq u_2$, then $p_{u_2} = 0$*

Proof. Given the same seed to draw Gaussian random points, let a set of random points from u_1 and u_2 be L_1 and L_2 respectively, the point d_1 where $d_1 = \arg\min_{d \in L_1} \sum_{j=1}^{j=m} w_j d[j]$, does not fall in S. This means $f(d_1) > \theta$. According to the same seed of generating data distribution, d_2 where $d_2 = \arg\min_{d \in L_2} \sum_{j=1}^{j=m} w_j \cdot d[j]$ must be dominated by d_1. According to Lemma 10, $\theta < f(d_1) < f(d_2)$, so we can conclude that none of $d \in L_2$ falls in S resulting in $p_{u_2} = 0$. □

Lemma 12. *Given 2 subspaces S_1 and S_2, if p_{u_i} evaluated in S_1 is equal to 0 and S_2 is a subspace of S_1, then p_{u_i} evaluated in S_2 is also 0.*

Proof. Let a set of random points from u be L, the point d where $d = \arg\min_{d \in L} \sum_{j=1}^{j=m} w_j d[j]$, does not fall in S_1 corresponding to $(\mathbf{w}, \theta_1)$. This means $f(d_i) > \theta_1$. We know that $S_2 \subseteq S_1$ where S_2 corresponds to $(\mathbf{w}, \theta_2)$, then $\theta_2 \leq \theta_1$. Because $f(d) > \theta_1 \geq \theta_2$, we can conclude that none of $d \in L$ falls in S_2 resulting in p_{u_2} evaluated in S_2 is 0. □

4.7.1 Cache-based evaluation

According to Lemmas 11 and 12, assuming that only one query is considered, that is, $\mathbf{w}$ is identical among all pairs, we conclude that if (u_1, θ_1) gives $p_{u_1} = 0$ then, for any (u_i, θ_i) that $u_1 \preceq u_i$ and $\theta_1 \geq \theta_i$, $p_{u_i} = 0$.

For this aim, every time we calculate p_{u_i} of (i, o_i), and its result is $p_{u_i} = 0$. That result can be a usable cache. We can keep or cache some potential (u_i, θ_i) to prune and reduce pairs in the next timestamp. Therefore, every timestamp especially in very first time, there are possible caches to be decided whether to keep for further usage.

However, we should carefully choose the number of caches and how to select caches. It is true that the more cache the more power to prune other pairs, but we cannot keep all caches. This is because each cache entry needs to compare with each object in U resulting in $O(N_C N_D)$ where N_C is the number of caches. This can be time-consuming as much as usual calculation of p_{u_i}. Moreover, in the case of failing to prune a tuple, possibly because of poor cache selection, that tuple eventually needs to be estimated by using a usual method that worsens calculation cost.

4.7.2 Cache selection

Actually, ideal caches which are able to prune a lot of other pairs should be a skyline. The definition of a skyline of a dataset is a set of points (objects), such that any point (object) is not dominated by any other point (object) in the dataset [54]. Therefore, an effective way to select caches is to choose points which are potentially close to the skyline. However, it is difficult and complex to perform skyline execution on a large set of caches which piles up every timestamp.

The research in [8] proposed an efficient sort-based method to calculate skyline. The key idea is to sort candidates based on scores calculated from a given monotonic function first and then calculate the skyline. An experiment shows a promising outcome

that the times of dominance comparisons can be significantly reduced because low-score candidates having high potential to prune other objects. Avoiding costly skyline execution, we intuitively keep the best potential caches by constructing a max-heap by letting the worst component at the top of the heap. When getting new cache candidates, we simply push them into the heap, and polling the worst cache (high score) out one by one until only the preferred size of caches left.

The score of a cache cannot be solely evaluated by its data point like the original sort-based skyline, but the timestamp and threshold must be taken into account. Caches with high thresholds, their data points that are supposed to be close to the origin and have smaller timestamp t' (high $\Delta t = t - t'$) are preferred because they have more pruning powers based on Definition 4.7.2, and Lemmas 11 and 12. Therefore, the following scoring function which takes these three factors into account is used to rank caches for cache selection.

$$score(u_i, \theta) = (1-\gamma)^{t-t'} \left[\sum_{i=1}^{m} (w_i d_i^{t'}[i]) - \omega\theta \right] \tag{4.8}$$

where $\gamma \in [0, 1)$ defines a decay factor of $score(u_i, \theta)$ w.r.t t, and $\omega \in [0, 1]$ is a weighting factor of the threshold. Caches with smaller scores calculated by using Eq.4.8 are more preferable. The outline of this method is described in Algorithm 9 which partially modified from Algorithm 8.

4.8 Simulation Experiments

A set of experiments were conducted by varying methods, parameters, datasets and computing environments to compare the performance of each solution.

4.8.1 Experiment setup

We ran the experiments on either a single node or a cluster of 10-node commodity PCs (Intel i5 1.4GHz/8GB, Ubuntu 12.04) connected via a Gigabit Ethernet switch. Apache Spark (v1.4) is installed on this cluster for parallel and distributed processing. The datasets are stored in a key-value distributed database–Apache HBase (v1.1.1). The programs were implemented using Java and run via Spark. In Spark, only the driver

Algorithm 9: Enhanced scheme: Cache-based method

Input: N_t, f, k, ε

Output: $T^0_{\varepsilon,k}, T^1_{\varepsilon,k}, \ldots, T^{N_t}_{\varepsilon,k}$

/* Same as lines 1--7 in Algorithm 8 */

1 $C \leftarrow$ an empty max-heap $(s, (u, \theta))$ for keeping caches using s as a key

2 $R \leftarrow \emptyset$

3 **for** *object* $u_i \in U$ **do**

4 **if** u_i *is not dominated by any* $c \in C$ **then**

5 $R \leftarrow R \cup (u_i, \theta)$

6 Calculate table T of $((1 - p_{u_i}), i)$ sorted by keys for each $(u_i, \theta) \in R$

7 $H \leftarrow$ an empty max-heap (p, u) using p as a key

8 **for** $(p_{u_i}, i) \in T$ **do**

9 **if** $(1 - p_{u_i}) = 1.00$ **then**

10 $C.insert(score(u_i, \theta), (u_i, \theta))$

11 **else if** $p_{u_i} t_p \geq \varepsilon$ **then**

12 $t_p \leftarrow p_{u_i} t_p$

13 **else**

14 $H.insert((1 - p_{u_i}), i)$

15 **while** $|C| > N_C$ **do**

16 $C.poll()$

/* Same as lines 29-34 in Algorithm 8 */

program (master node) can request data tuples from HBase while the rest of nodes only help computation. Technically the total execution time not only includes computation time but also database access time, communication and data transfer time in the cluster.

In each experiment, we simulated a single continuous query. Some default parameters (*e.g.*, τ, γ, ω) were chosen based on parameter studies. We chose moderate values that provide satisfactory outcomes for general cases. We found that the performance on varying some parameters can fluctuate but not to the degree that significantly changes the interpretation. The query specification and default parameters as well as their ranges in the experiments are shown in Table 4.2.

Table 4.2: Parameters in the experiments

Parameter	Range/Value	(Default)
f	$\mathbf{w} = \{0.2, 0.2, 0.6\}$	-
k	30	-
N_T	500	-
ε	[0.1, 1.0]	(0.8)
N_S	[1000, 50k]	(10k)
τ in Eq.4.5	[0.25, 2.0]	(1.0)
γ in Eq.4.8	0.2	-
ω in Eq.4.8	0.2	-

4.8.2 Datasets

We used two real-world multi-dimensional time-series datasets. The attributes to be used were selected based on commonness, completeness of data, and types of data (qualitative measures) as follows:

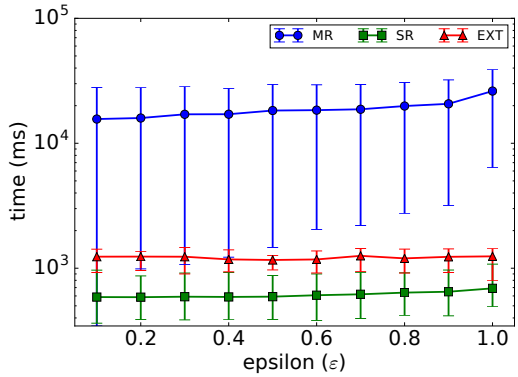
1. **NOAA dataset**[5] This dataset contains hourly climate normal data over years from 458 US weather stations. We selected 3 attributes including temperature (temp), dew point (dewp) and average wind speed (wavgspd). There are 8760 timestamps, but we split the first 1000 snapshots for estimating the variances of each attribute according to the method in Section 4.4.3.

2. **Weathernews dataset**[6] This is also climate data updated every 15 minutes from 821 weather stations in Japan. We selected 3 attributes including temperature (temp), wind speed (windspd) and precipitation rate (prec10). In the same way, from available 30000 snapshots, we split the first 1000 snapshots for estimating the variances of each attribute according to the method in Section 4.4.3.

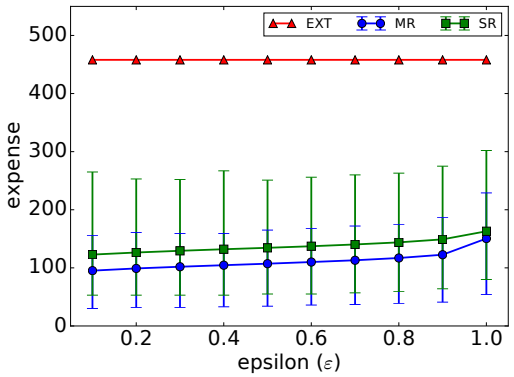
Because the synthetic datasets with moving objects/values are usually modeled based on Gaussian random walk which is similar to our underlying assumption of our methods, we decided to solely show the results of the real datasets without bias.

[5] http://www.ncdc.noaa.gov/

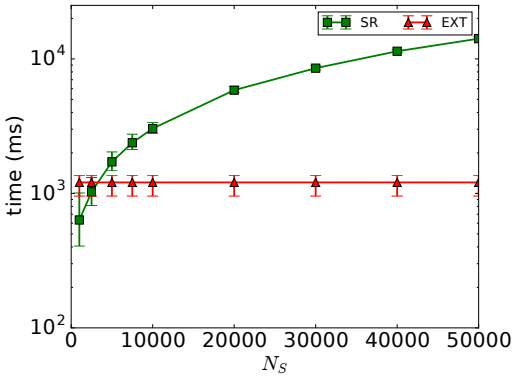
[6] https://labs.weathernews.jp/data.html



(a) Varying ε against time - showing high calculation time of the MR method



(b) Varying ε against expense - showing the constantly high expense of the EXT method compared with the MR and SR methods



(c) Varying N_S against time - showing that the calculation time of the proposed SR method increases significantly with the number of samplings N_S

Figure 4.4: NOAA: Results of each method on a single node

4.8.3 Methods

1. **Exact method (EXT)** calculates exact T_k^t by retrieving D^t using a single machine

2. **Multiple-round proposed method (MR)** calculates $T_{\varepsilon,k}^t$ as explained in Section 4.5.1

3. **Single-round proposed method (SR)** calculates $T_{\varepsilon,k}^t$ as explained in Section 4.5.2

4. **Cache-based single-round method (CSR)** calculates $T^t_{\varepsilon,k}$ as explained in Section 4.7

4.8.4 Benchmarks

1. **Expense:** This metric counts the number of data tuple requests to the key-value store database. Our objective is to minimize this metric as much as possible while other metrics should be kept reasonably good.

2. **Accuracy:** Being able to reduce expenses, the quality of the result set cannot be compromised. This metric shows the correctness of the final top-k answers in each timestamp. Because a top-k query is a rank-aware query, using precision which takes the result as an unordered set may not be appropriate. Thus, we use a variant of Kendall tau distance [25] to indicate the degree of ordering difference (pairwise distinct) between a given result and the exact result. Given approximate answer set $T^t_{\varepsilon,k}$ and the exact answer set T^t_k, the accuracy is defined as

$$R(T^t_{\varepsilon,k}, T^t_k) = 1 - \frac{\sum_{\{i,j\}\in P} K'_{i,j}(r_1, r_2)}{|P|} \tag{4.9}$$

where P is the set of unordered pairs of elements in T^t_k [7], r_1 and r_2 are the rankings of objects in $T^t_{\varepsilon,k}$ and T^t_k respectively and

$$K'_{i,j}(r_1, r_2) = \begin{cases} 0 & \text{if } i,j \text{ are in the same order in both lists} \\ 1 & \text{otherwise.} \end{cases}$$

In our experiments, $R = 1$ means that the answers in T^t_k and $T^t_{\varepsilon,k}$ are identical while $R < 1$ stated that $T^t_k \backslash T^t_{\varepsilon,k} \neq \emptyset$. Given that $T^t_k = \{o_1, o_2, o_3\}, T_A = \{o_4, o_1, o_3\}$ and $T_B = \{o_1, o_2, o_5\}$, it is noted that $R(T_A, T^t_k) < R(T_B, T^t_k)$ because the irrelevant answer o_4 replaces the first rank while o_5 replaces the third rank which is less severe.

3. **Execution time:** The execution time to calculate T^t_k or $T^t_{\varepsilon,k}$ in each timestamp was measured by wall-clock time in milliseconds (ms). Since the execution time can fluctuate on each timestamp, we generally show the average case except the range (minimum, maximum) indicated otherwise.

[7] The definition here is a little different from the original Kendall tau distance in [25]

4.8.5 Results of NOAA dataset

Trade-off in each method

In this experiment, we test 3 methods including the EXT, MR and SR methods on a single node to examine their trade-off.

For the MR and SR methods, only in this experiment, we set $N_S = 1000$ for comparison purposes, in fact, this is rather small. In Fig.4.4a, we varied ε and measured the execution time for each timestamp in each method. The graph shows the average execution time while the upper and lower bars show the maximum and minimum execution time ever recorded in each timestamp. We obviously see that calculating $T^t_{\varepsilon,k}$ in the MR method is more costly than calculating T^t_k in the EXT method because the MR method carefully retrieves a data tuple at a time and re-evaluate t_p every time resulting in significantly higher execution time. While the SR method requires a calculation of t_p only one time per snapshot and N_S is small, in this case, the SR method can run even faster than the EXT method. However, the advantages of the MR and SR methods can be apparently seen in Fig.4.4b that the expense can be largely reduced compared with the EXT method. Regarding the quality of the answers (accuracy), we will discuss next.

Although the MR method can save more expenses than other methods, due to the higher order of magnitude of computation time, it is impractical when N_S gets higher. Therefore, in what follows, we omit the results of the MR method as we already know that it outperforms the SR method in terms of expense. The SR method which is derived from the MR method was used by default for the rest of the experiments.

Fig.4.4c shows that the average execution time of the SR method significantly grows with the number of samplings (N_S). This emphasized that calculating t_p (even a single round) is a major computing-intensive task.

Expense analysis

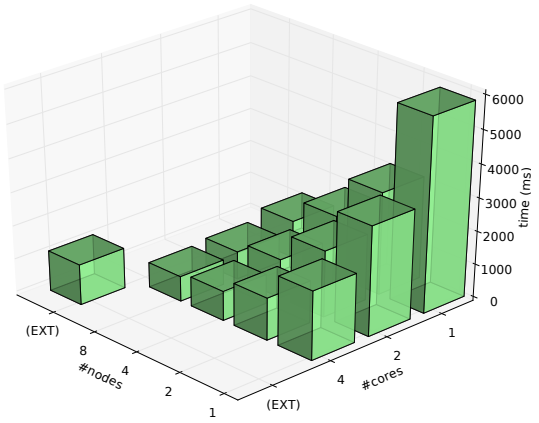
Increasing the confidence factors, *i.e.*, ε and τ, definitely results in higher expense because many data tuples need to be accessed to ensure the quality of the answers. In this experiment, we show the result of the accuracy of $T^t_{\varepsilon,k}$ as well as its expense by varying ε and τ. Table 4.3 shows the minimum accuracy and the average accuracy respectively in each cell, while Table 4.4 shows the expense and the expense reduction percentage from

Table 4.3: NOAA: The minimum/average accuracy of the SR method - showing that, at the default setting ($\varepsilon = 0.8, \tau = 1.0$), the proposed SR method can achieve promising accuracy while increasing ε to 1.0 can give perfect accuracy

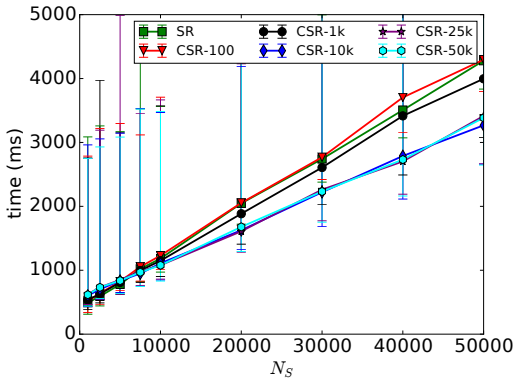
ε/τ	**0.25**	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
0.1	0.634/0.879	0.747/0.976	0.869/0.988	0.933/0.999	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000
0.2	0.634/0.886	0.807/0.979	0.869/0.992	0.933/1.000	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.3	0.634/0.891	0.807/0.980	0.869/0.994	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.4	0.690/0.900	0.807/0.981	0.869/0.996	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.5	0.690/0.902	0.807/0.977	0.869/0.997	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.6	0.690/0.916	0.807/0.979	0.933/0.998	0.933/0.999	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.7	0.690/0.924	0.807/0.979	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.8	0.690/0.933	0.807/0.983	0.933/1.000	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.9	0.690/0.952	0.869/0.990	0.933/1.000	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000
1.0	0.807/0.977	0.933/0.999	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000

Table 4.4: NOAA: The expense of the SR method and its reduction percentage from the EXT method - showing that, at the default setting ($\varepsilon = 0.8, \tau = 1.0$), the proposed SR method can cut the expense by 68%

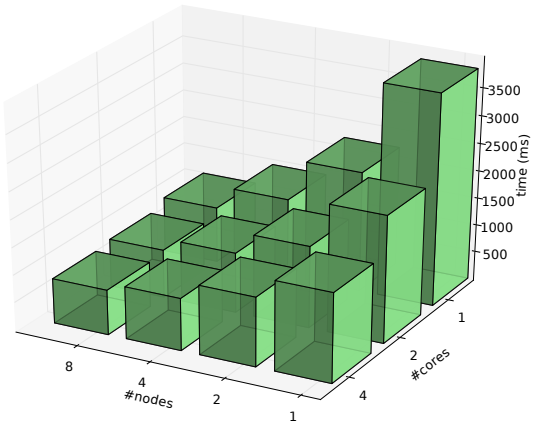
ε/τ	**0.25**	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
0.1	38139 (-83%)	46988 (-79%)	55118 (-76%)	62494 (-73%)	69336 (-70%)	75578 (-67%)	81468 (-64%)	87193 (-62%)
0.2	38679 (-83%)	48062 (-79%)	56646 (-75%)	64383 (-72%)	71516 (-69%)	77994 (-66%)	84282 (-63%)	90324 (-60%)
0.3	39128 (-83%)	48925 (-79%)	57866 (-75%)	65889 (-71%)	73198 (-68%)	79906 (-65%)	86417 (-62%)	92747 (-59%)
0.4	39630 (-83%)	49790 (-78%)	58977 (-74%)	67228 (-71%)	74708 (-67%)	81671 (-64%)	88416 (-61%)	94873 (-58%)
0.5	40066 (-82%)	50553 (-78%)	60057 (-74%)	68550 (-70%)	76214 (-67%)	83422 (-63%)	90400 (-60%)	96988 (-58%)
0.6	40563 (-82%)	51418 (-77%)	61182 (-73%)	69934 (-69%)	77778 (-66%)	85261 (-63%)	92518 (-60%)	99168 (-57%)
0.7	41084 (-82%)	52393 (-77%)	62446 (-73%)	71471 (-69%)	79581 (-65%)	87381 (-62%)	94854 (-58%)	101647 (-56%)
0.8	41726 (-82%)	53575 (-77%)	64068 (-72%)	73453 (-68%)	81886 (-64%)	90092 (-61%)	97748 (-57%)	104643 (-54%)
0.9	42791 (-81%)	55451 (-76%)	66531 (-71%)	76363 (-67%)	85359 (-63%)	94129 (-59%)	101997 (-55%)	109111 (-52%)
1.0	49268 (-78%)	66243 (-71%)	80603 (-65%)	93795 (-59%)	105640 (-54%)	116119 (-49%)	126273 (-45%)	135156 (-41%)



(a) The SR and EXT methods against time - showing that increasing the number of cores and the number of nodes in the SR method can significantly reduce the total execution time



(b) Varying N_S against time - showing the total execution time of the CSR method with n caches (CSR-n) and the SR method



(c) The CSR method (CSR-10k) against time - showing the less total execution time compared with Fig.4.5a

Figure 4.5: NOAA: Results of each method on a cluster

the constant expense of the EXT method in each cell. The results show that even setting $\varepsilon = 0.1$ and $\tau = 0.5$ still give promising average accuracy at 0.879, and the expense can be largely reduced by 83%. In the case of our default setting ($\varepsilon = 0.8, \tau = 1.0$), even though the minimum accuracy is not 1 but its average accuracy (in 3 decimals) is close to 1. In addition, the expense is suppressed by 68%. If we increase $\varepsilon = 0.8$ to 1.0, the result shows the perfect accuracy that all $T^t_{\varepsilon,k}$ is exactly the same as T^t_k while

the expense is more than halved (-59%).

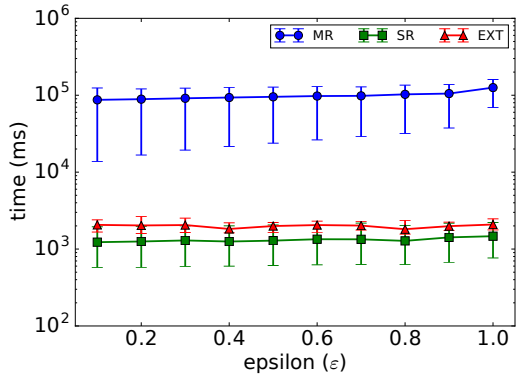
It is noted that, for applications that require high precision, setting extremely-high parameters $\varepsilon = 1.0$ and $\tau = 2.0$ which also give the perfect results can still reduce the expense up to 41%. This is obviously better than paying the cost to access all data tuples to calculate T_k^t in the EXT method.

In-cluster Deployment

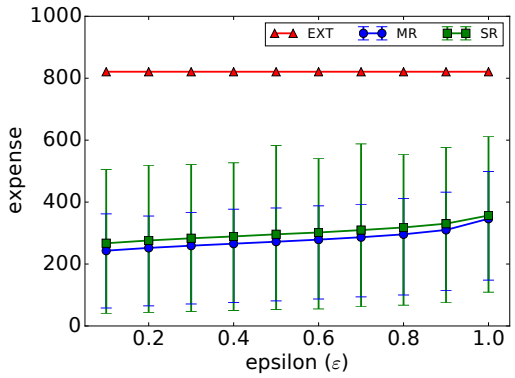
We implemented a variant of our proposed SR method to compute in a parallel and distributed fashion in Hadoop (via Spark). Fig.4.5a shows the average execution time in each timestamp by varying the number of nodes and the number of cores in each node. The number of virtual cores is the product of the number of nodes and the number of cores. The isolated bar (EXT) shows the average execution time of the EXT method computed in a local node (1 node). We can see that increasing the number of nodes or the number of cores can speed up the task even though the improvement may not be linear with the number of virtual cores. At higher than 8 virtual cores, the average execution time is lower than that of the EXT method. This is because the SR method can reduce the number of data accesses resulting in lower total data access time while the expensive execution time of probability estimation can be relived by parallel computing. This is important because, thinking about accessing external data sources which need transferring data requests via the Internet unlike a distributed database in the same network in this experiment, in that case, it yields a huger gap in terms of time latency. This confirms the effectiveness of our proposed method which reduces expenses bringing about more computing complexity, because, with existing computing resources, the execution time can be relieved. Furthermore, it can further reduce the uncontrollable latencies from data access and possibly becomes faster than the traditional computation.

Performance of the enhanced method

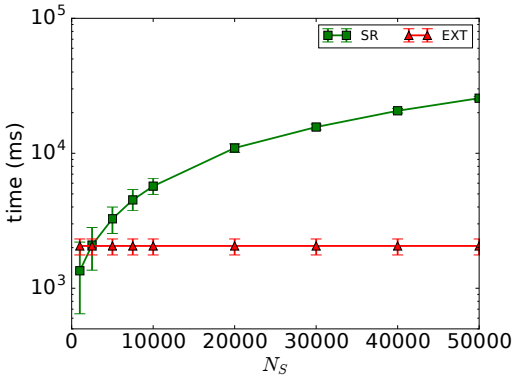
Fig.4.5b shows the results of the SR method and the CSR method with n caches (CSR-n). The result shows that increasing the number of samplings, the average execution time tends to increase linearly. We can see that using only a small number of caches, i.e, CSR-100, CSR-1000, at high N_S, it cannot even outperform the baseline SR method. This is because a small number of caches cannot prune a lot of candidates while the CSR



(a) Varying ε against time - showing high calculation time of the MR method



(b) Varying ε against expense - showing the constantly high expense of the EXT method compared with the MR and SR methods



(c) Varying N_S against time - showing that the calculation time of the proposed SR method increases significantly with the number of samplings N_S

Figure 4.6: WN: Results of each method on a single node

method also incurs additional computation for cache maintenance (*i.e.*, cache ordering and cache selection). While using more caches, *i.e.*, CSR-10k, CSR-25k and CSR-50k, in small N_S is worse than the others due to large overhead of cache maintenance. In fact, it can prune a lot of unnecessary calculation tasks. However, when N_S gets higher, the benefit of using caches over not using caches or using a small number of caches can be obviously seen. For the number of caches that is higher than 10k, their performance is not significantly different. Therefore, we conclude that we should use a sufficiently

high number of caches to efficiently reduce the computation cost. Anyway, this also depends on the number of attributes (dimensionality) and data distribution.

In addition, Fig.4.5c shows the execution time of the CSR method (CSR-10k) in a cluster on various numbers of nodes and cores. Increasing the number of nodes or cores can speed up the computation time. However, it is noted that the speed up ratio gradually becomes not proportional to the number of virtual cores when increased.

4.8.6 Result of WN dataset

We also conducted experiments on the WN dataset by using the same environments as Section 4.8.5. The WN dataset has the same dimensionality and larger number of objects compared with the NOAA dataset.

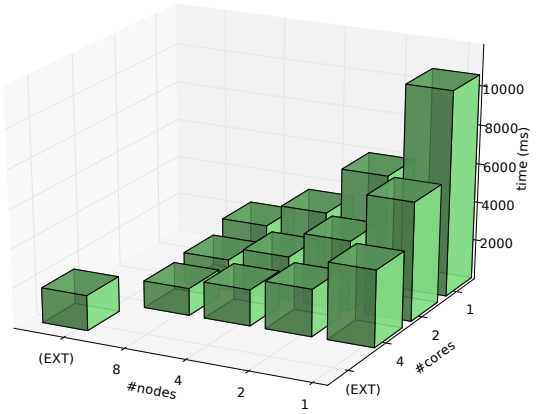
Trade-off in each method

Similar to the experiments of the NOAA dataset, N_S was set at 1000 for the MR and SR methods. Fig.4.6a shows the result obtained by measuring the average execution time on various ε. The average execution time is higher than that of the NOAA dataset because of the higher number of data objects. However, it appears that the result is consistent with that of the NOAA dataset, *i.e.*, the MR method performed worse than the other 2 methods significantly.

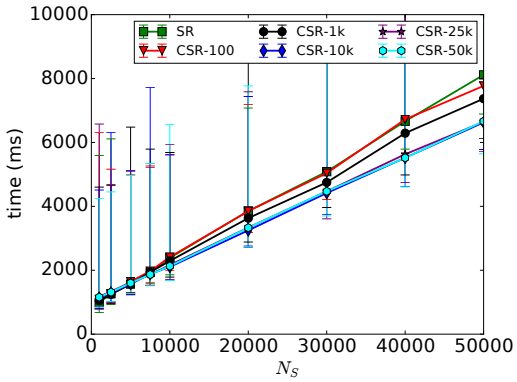
In terms of expense, as shown in Fig.4.6b, the SR and MR methods request data objects significantly less than the EXT method, which constantly retrieves $N_D = 821$ objects in each timestamp. However, it seems that the difference of expense between the SR method and the MR method is trivial while the execution time of the MR method is higher than that of the SR method more than an order of magnitude. Again, in what follows, we omit the results of the MR method and used the SR method as a default scheme. Fig.4.6c shares the same characteristics with Fig.4.4c that the execution time in the SR method increases rapidly as increasing the number of samplings.

Expense analysis

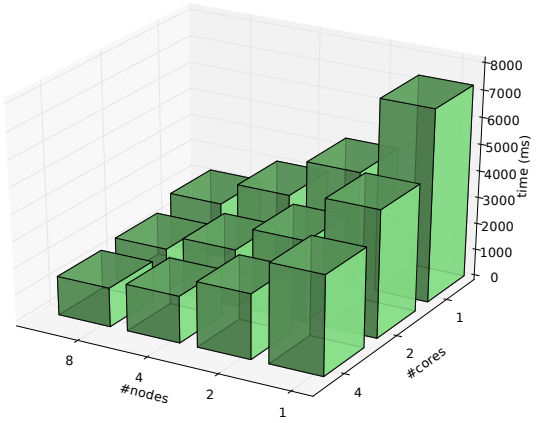
The outcomes of the accuracy and the expense from this dataset shown in Table 4.5 and Table 4.6 respectively are slightly poorer than that of the NOAA dataset. In our default



(a) The SR and EXT methods against time - showing that increasing the number of cores and the number of nodes in the SR method can significantly reduce the total execution time



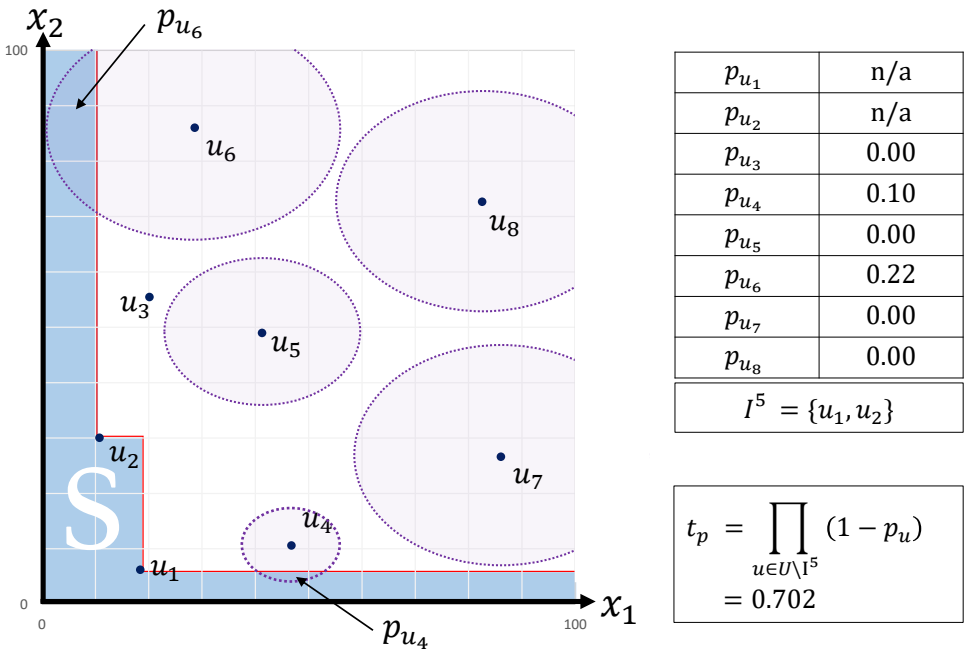
(b) Varying N_S against time - showing the total execution time of the CSR method with n caches (CSR-n) and the SR method



(c) The CSR method (CSR-10k) against time - showing the less total execution time compared with Fig.4.7a

Figure 4.7: WN: Results of each method on a cluster

setting ($\varepsilon = 0.8, \tau = 1.0$), the minimum accuracy and the average accuracy are recorded close to the NOAA dataset at 0.933 and 0.999 respectively while the expense can be reduced by 59% which is 9% less than that of the NOAA dataset. If $\varepsilon = 0.8$ is increased to 1.0, the result also shows the perfect accuracy as same as the NOAA dataset while the expense is suppressed by 49%. Setting extremely-high confidence factors $\varepsilon = 1.0$ and $\tau = 2.0$ which definitely gives the perfect results can lessen expense up to 26%.



p_{u_1}	n/a
p_{u_2}	n/a
p_{u_3}	0.00
p_{u_4}	0.10
p_{u_5}	0.00
p_{u_6}	0.22
p_{u_7}	0.00
p_{u_8}	0.00

$I^5 = \{u_1, u_2\}$

$$t_p = \prod_{u \in U \setminus I^5} (1 - p_u) = 0.702$$

Figure 4.8: The illustration of uncertain objects in Table 4.1 (Skyline)

This again confirms the advantages of our method over the EXT method.

In-cluster deployment and the enhanced method

Fig.4.7a shows the same phenomenon as Fig.4.5a that the expensive calculations in the SR method can be alleviated by using a cluster. Again, due to the reduction of data requests, it can eventually outperform the EXT method in terms of total execution time.

In Fig.4.6c, the result is in accordance with Fig.4.4c that using the number of caches around 10k-50k gives better result when $N_S > 10k$. Even if the gain of using the CSR method over the SR method seems small when N_S is small, *e.g.*, $N_S = 7500$, this figure shows the execution time per timestamp. Therefore, in real monitoring system/analytic calculation, many timestamps need to be computed resulting in an apparent difference between methods. Moreover, the performance of the CSR method can be improved by adding more nodes as shown in Fig.4.7c.

Table 4.5: WN: The minimum/average accuracy of the SR method - showing that, at the default setting ($\varepsilon = 0.8, \tau = 1.0$), the proposed SR method can achieve promising accuracy while increasing ε to 1.0 can give perfect accuracy

ε/τ	**0.25**	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
0.1	0.582/0.897	0.747/0.972	0.869/0.990	0.869/0.996	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000
0.2	0.634/0.903	0.807/0.975	0.869/0.991	0.869/0.997	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000
0.3	0.690/0.910	0.807/0.977	0.869/0.993	0.869/0.997	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000
0.4	0.690/0.916	0.807/0.979	0.869/0.993	0.933/0.999	0.933/1.000	0.933/1.000	1.000/1.000	1.000/1.000
0.5	0.690/0.918	0.807/0.983	0.869/0.994	0.933/0.999	0.933/1.000	0.933/1.000	1.000/1.000	1.000/1.000
0.6	0.690/0.921	0.807/0.983	0.869/0.995	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.7	0.690/0.933	0.869/0.986	0.869/0.996	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.8	0.747/0.937	0.807/0.987	0.869/0.996	0.933/0.999	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000
0.9	0.747/0.946	0.807/0.990	0.869/0.997	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000
1.0	0.807/0.973	0.869/0.996	0.933/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000	1.000/1.000

Table 4.6: WN: The expense of the SR method and its reduction percentage from the EXT method - showing that, at the default setting ($\varepsilon = 0.8, \tau = 1.0$), the proposed SR method can cut the expense by 59%

ε/τ	**0.25**	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
0.1	67910 (-83%)	95535 (-77%)	119033 (-71%)	139676 (-66%)	158378 (-61%)	175131 (-57%)	190510 (-54%)	204614 (-50%)
0.2	69688 (-83%)	98411 (-76%)	122853 (-70%)	144303 (-65%)	163569 (-60%)	180902 (-56%)	196665 (-52%)	211160 (-49%)
0.3	71147 (-83%)	100785 (-75%)	125886 (-69%)	147879 (-64%)	167659 (-59%)	185314 (-55%)	201439 (-51%)	216233 (-47%)
0.4	72426 (-82%)	102794 (-75%)	128610 (-69%)	151147 (-63%)	171268 (-58%)	189345 (-54%)	205749 (-50%)	220829 (-46%)
0.5	73618 (-82%)	104952 (-74%)	131218 (-68%)	154386 (-62%)	174845 (-57%)	193277 (-53%)	209990 (-49%)	225326 (-45%)
0.6	74837 (-82%)	107074 (-74%)	134128 (-67%)	157746 (-62%)	178634 (-56%)	197412 (-52%)	214422 (-48%)	229993 (-44%)
0.7	76384 (-81%)	109638 (-73%)	137337 (-67%)	161564 (-61%)	182923 (-55%)	202086 (-51%)	219495 (-47%)	235269 (-43%)
0.8	78301 (-81%)	112643 (-73%)	141296 (-66%)	166279 (-59%)	188211 (-54%)	207868 (-49%)	225718 (-45%)	241809 (-41%)
0.9	81125 (-80%)	117280 (-71%)	147340 (-64%)	173405 (-58%)	196170 (-52%)	216502 (-47%)	234775 (-43%)	251508 (-39%)
1.0	96927 (-76%)	142406 (-65%)	179578 (-56%)	210877 (-49%)	237923 (-42%)	261827 (-36%)	283005 (-31%)	302013 (-26%)

4.9 Extension to Skyline Monitoring

In this section, the cost-minimizing techniques for skyline monitoring called epsilon skyline processing (ε-skyline processing) which is extended from the same foundation of ε-top-k processing is proposed. For definitions and details of skyline, please refer to Chapter 3.

Fig.4.8 shows the example of uncertain objects from Table 4.1 with various uncertainties similar to Fig.4.2 and the skyline execution over those uncertain objects.

Definition 4.9.1. An ε-skyline query returns the up-to-date answer set S_ε^t that, based on a given uncertain model, with the probability higher than ε that S_ε^t calculated from U is not different from S^t calculated from D^t.

In the case of ε-skyline query processing, to calculate p_{u_i} for each $u_i \in U$ as well as t_p can use Eq.4.6 and Eq.4.7 respectively by defining space S as the area that is not dominated by the intermediate answers (u_1 and u_2 in Fig.4.8).

Given I^t be the intermediate answers, I^t is acceptable as the approximate skyline answers w.r.t ε if and only if $t_p \geq \varepsilon$. Otherwise, the same strategy to increase t_p, *i.e.*, eliminate some uncertainties, as described in Section 4.4.4 can be used until t_p reaches above ε.

4.9.1 Expense analysis

Both 2 approaches for ε-top-k processing including single-round method (SR) and multiple-round method (MR) can be re-implemented for calculating S_ε^t. Because the main objective is to reduce the expense, we analyze and report only the results regarding accuracy and expense by using the WN dataset (2 attributes including temperature and wind speed), and, in this analysis, the single-round method is chosen because of less and reasonable execution time.

Table 4.7 shows the minimum accuracy and the average accuracy evaluated by F-measure (F1) [60]. It is noted that this evaluation metric is different from Section 4.8.4 because the result set of skyline is unordered. Table 4.8 shows the expense and the expense reduction percentage from the constant expense of the EXT method in each cell.

Setting $\varepsilon = 1.0$ and $\tau = 1.0$ can record the perfect accuracy while the expense can be reduced up to 61%. In the case that some applications can tolerate some errors, the choices of lower ε can yield more saving expense.

4.10 Conclusions

Prior works [48, 56, 66] have projected that massive sensor data generated from various fields will be collected, managed, and utilized under an umbrella of cloud-based systems called sensor cloud. In this work, we proposed a cost-minimizing framework for top-k monitoring in sensor cloud services. Modeling aged sensor data with uncertainty, the proposed ε-top-k monitoring method can save cost by avoiding unpromising data requests while the quality of the results can be kept very high. We have confirmed the effectiveness of our framework through extensive experiments on real-world datasets. In addition, this paper has demonstrated the better computing performance of our framework when running on well-known Hadoop which is popular to be deployed in the cloud as well as the enhanced scheme to further decrease the computation time. The experiment results provided compelling results that, using distributed computing on such system, the computation time and the data access time of our more complex methods can be largely reduced and can be even faster than the conventional less complex method due to fewer data requests. In addition, these methods can be easily adapted to work with skyline monitoring as explained in Section 4.9.

This work is the first to consider a critical cost when processing continuous queries in such environment. As the use of cloud computing for monitoring purpose becomes more widespread, we have shown that our proposed framework can help users wisely utilize their budget (more economical). However, there are some limitations, for example, non-stationary data – possibly cannot reduce data accesses much by the assumed model and the necessity to make parametric models as well as perform parameter studies in advance.

Table 4.7: WN2d: The minimum/average accuracy of the SR method (Skyline) - showing that setting $\varepsilon = 1.0$ and $\tau = 1.0$ can achieve can give perfect accuracy

ε/τ	**0.25**	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
0.1	0.429/0.941	0.667/0.980	0.727/0.992	0.727/0.996	0.727/0.998	0.889/1.000	0.933/1.000	0.889/1.000
0.2	0.500/0.951	0.750/0.983	0.727/0.994	0.727/0.997	0.727/0.998	0.889/1.000	0.933/1.000	1.000/1.000
0.3	0.500/0.953	0.667/0.983	0.727/0.995	0.727/0.997	0.727/0.999	0.933/1.000	0.933/1.000	0.727/0.999
0.4	0.462/0.953	0.667/0.984	0.727/0.995	0.727/0.997	0.889/0.999	0.933/1.000	0.889/1.000	0.889/1.000
0.5	0.500/0.957	0.667/0.984	0.727/0.995	0.727/0.997	0.889/0.999	0.933/1.000	0.889/1.000	0.889/1.000
0.6	0.500/0.958	0.750/0.985	0.727/0.996	0.727/0.998	0.933/1.000	0.933/1.000	0.889/1.000	0.889/1.000
0.7	0.545/0.961	0.750/0.986	0.727/0.996	0.727/0.998	0.933/1.000	0.933/1.000	0.727/0.999	0.889/1.000
0.8	0.667/0.966	0.750/0.986	0.727/0.996	0.727/0.998	0.933/1.000	0.889/1.000	0.889/1.000	0.889/1.000
0.9	0.625/0.967	0.727/0.991	0.727/0.997	0.889/0.999	0.933/1.000	0.889/1.000	0.889/1.000	1.000/1.000
1.0	0.750/0.984	0.727/0.996	0.933/1.000	1.000/1.000	0.889/1.000	1.000/1.000	1.000/1.000	1.000/1.000

Table 4.8: WN2d: The expense of the SR method (Skyline) and its reduction percentage from the EXT method - showing thatsetting $\varepsilon = 1.0$ and $\tau = 1.0$ can cut the expense by 61%

ε/τ	**0.25**	**0.50**	**0.75**	**1.00**	**1.25**	**1.50**	**1.75**	**2.00**
0.1	28421 (-93%)	50594 (-88%)	72069 (-82%)	90197 (-78%)	107461 (-74%)	124939 (-70%)	139732 (-66%)	153753 (-63%)
0.2	29717 (-93%)	53035 (-87%)	75524 (-82%)	94370 (-77%)	112239 (-73%)	130687 (-68%)	145898 (-64%)	160634 (-61%)
0.3	30755 (-93%)	54955 (-87%)	78029 (-81%)	97875 (-76%)	116188 (-72%)	134755 (-67%)	150674 (-63%)	166652 (-59%)
0.4	31706 (-92%)	56646 (-86%)	80386 (-80%)	100913 (-75%)	120522 (-71%)	138540 (-66%)	154714 (-62%)	171841 (-58%)
0.5	32794 (-92%)	58436 (-86%)	82750 (-80%)	103836 (-75%)	124798 (-70%)	142380 (-65%)	159085 (-61%)	176474 (-57%)
0.6	33871 (-92%)	60276 (-85%)	85237 (-79%)	106974 (-74%)	128775 (-69%)	146648 (-64%)	164504 (-60%)	181204 (-56%)
0.7	35038 (-91%)	62330 (-85%)	88113 (-79%)	110420 (-73%)	132717 (-68%)	151210 (-63%)	170502 (-58%)	186494 (-55%)
0.8	36524 (-91%)	64902 (-84%)	91825 (-78%)	114594 (-72%)	137588 (-66%)	156636 (-62%)	176968 (-57%)	193036 (-53%)
0.9	39210 (-90%)	70592 (-83%)	97628 (-76%)	123367 (-70%)	145386 (-65%)	167095 (-59%)	186146 (-55%)	204336 (-50%)
1.0	52524 (-87%)	93101 (-77%)	129244 (-69%)	159588 (-61%)	189099 (-54%)	215084 (-48%)	238054 (-42%)	261696 (-36%)

Chapter 5

Summary

5.1 Summary of Contributions

In the era of enormous data generated every second by prevalent sensors and mobile devices as often stressed in the contexts of Big Data and Internet of Things (IoT), we have illustrated that a number of crucial monitoring applications to prevent undesirable incidents require real-time data processing to acquire useful insights. Continuous preference query processing is an important part to achieve this by taking roles of processing those data and delivering only a small number of insightful data to the right person. However, data management systems to handle massive data as well as data sources are likely to be more distributed, *e.g.*, sensor networks and cloud, rather than centralized.

The principal objective of this thesis is to propose solutions towards the problems of Continuous preference query processing in such distributed environments. In this thesis, we mainly focus on two main fundamental queries including top-k queries and skyline queries while executing these queries in different environment contexts has different limitations and different techniques to alleviate the limitations.

In traditional highly distributed systems, where the amount of data transfer is critical, we propose a subscription-based continuous top-k query processing method. Instead of aggregating all dynamic data records at the base station, using subscriptions bound to sources of data generation to indicate preferable data can intuitively reduce the large amount of data records to be transferred, but the direct implementation can be non-scalable in terms of the number of concurrent users because the communication

cost of subscription dissemination is non-trivial. In Chapter 2, we eliminated this weakness of this scheme by pointing out that not all subscriptions need to be disseminated, but instead only dissemination of our proposed small set of minimal subscriptions is enough for this purpose without sacrificing the correctness. In addition, we can avoid sending subscriptions to some unimportant nodes to further reduce more communication cost too [79]. The algorithms for this processing as well as the strategic approaches to maintain subscriptions have been elaborated.

In Chapter 3, we focused on another preference query called skyline query processing [10]. Skyline calculation is a compute-intensive task (expensive computation cost). When this query needs to be computed repeatedly for monitoring purposes, it can take huge amount of time which finally reflects the response time. Hence, we proposed an algorithm for continuous skyline query processing on frequent data streams aiming at speeding up skyline computation at the centralized server, *i.e.*, the base station. Our proposed method derives benefits from using minimum bounding rectangles and their properties to summarize each data object's changes in each snapshot and identify a smaller set of candidates for skyline computation respectively. The advantage using this approach is that the pruning strategy is adaptive based on data distribution and update patterns. As a result, through the experiments, our proposed method shows the better results in terms of total execution time over the other methods.

In Chapter 4, we focus on cost-efficient approaches to process preference queries for more abstracted architectures, such as sensor cloud. In such systems, to acquire data is charged by the number of data requests which is costly for continuous preference query processing. We proposed a variant of a top-k query called ε-top-k query which returns an approximate result set with controllable errors. The empirical experiments on the real datasets show that the cost of data access (expense) using our proposed method can be significantly minimized while the accuracy can be preserved at a satisfactory level. This idea can be extended and applied to skyline query processing, and the obtained results are also promising as that of top-k query processing. Despite calculation complexity of our proposed query, using state-of-the-art distributed computing frameworks (for instance MapReduce and Spark) and our proposed techniques to avoid unnecessary calculations can accelerate the computation time and even faster than the conventional method (due to the less total latency of data requests).

In summary, continuous preference query processing helps people make decisions

and take actions on massive real-time data which are usually managed in distributed environments. Our proposed solutions in this thesis try to minimize the concerned costs in each architecture, for example, communication cost which affects the network lifetime in traditional schemes of distributed systems, computation cost which affects response time and data access cost which reflects expense to users. Therefore, with the same architectures, our proposed methods improve scalability – able to handle more queries and more data as well as network lifetime – able to perform monitoring queries longer. While, in the case of the cost of data access, with the same amount of budget, our proposed methods make monitoring queries can run even longer than not using it.

5.2 Future Work

Through this thesis, we found the following remaining issues open to our future work.

Monitoring queries on complex data

This thesis mainly focused on two fundamental queries on multi-attribute dataset (multidimensional data), which are related to the field of multi-criteria decision making. However, there are demands for other complex query processing on different data types too, such as graph data and text. For example, social network monitoring, which observes community structures as well as content produced by the community, plays an important role on understanding community characteristics and features. A marketing person may want to monitor the evolution of the community structure (graph data) by time to analyze meaningful users' behaviors, *e.g.*, k-core communities [55, 63], interests with large communities [77] and top-k influential nodes [90, 99]. In addition, he may want to monitor on the development of the content (text data) produced by communities to understand the drift of their interests (evolving concepts and topics). Seeing that graph data can be represented by a huge matrix and text data can be represented by very high-dimensional vectors, to put these monitoring queries in real-practice is challenging because of its intense data dynamicity, enormous size and high calculation complexity. A future direction on this research topic should be focused on algorithmic improvements to enable to deliver up-to-date results in time.

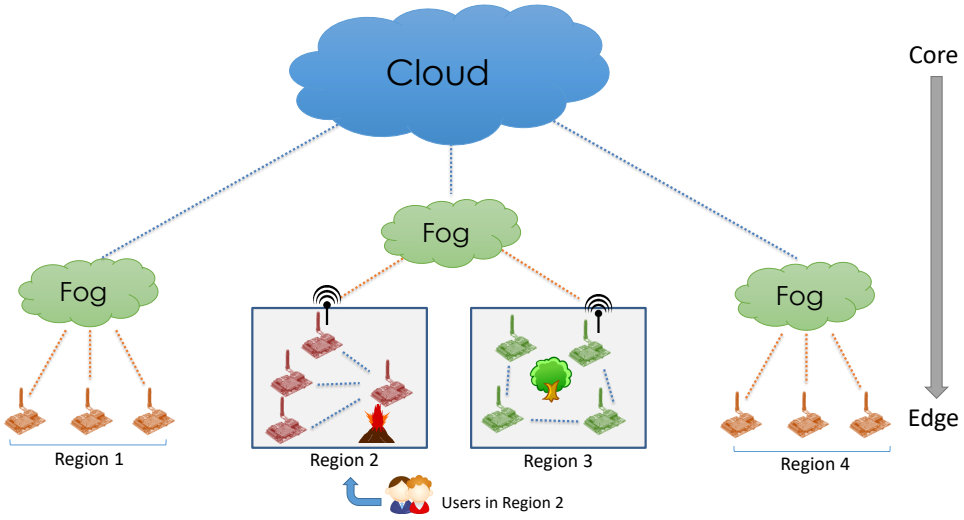


Figure 5.1: Cloud, Fog and Edge computing

Cloud, Fog and Edge computing and their cost models

In this thesis, we consider continuous query processing on two main paradigms of distributed environments when dealing with sensor data, *i.e.*, traditional highly distributed systems (*e.g.*, sensor networks, P2P systems) and cloud-extended systems (*e.g.*, sensor cloud). According to Fig.5.1, those can be compared as edge computing (some processes done at local nodes) and cloud computing (processes at the core) respectively. Processing continuous queries for such both cases have different cost models as discussed previously in Chapter 1 – 4.

Recently, fog computing has been a buzzword introduced in 2012 [47] for being a smaller cloud that is put away from the centralized point (*e.g.*, cloud) and closer to the edge (*e.g.*, sensors, mobile devices, and also users) as illustrated in Fig.5.1. This claims benefits of better response time on location-aware applications where users or clients and data are in proximity because a large amount of data can be served by a fog instead of accessing data to the more expensive main cloud as a last resort,

For continuous query optimization, it is important to identify the cost model for such mixed paradigm which is definitely unlike what it has been discussed in this thesis. Also, the problem on how to efficiently execute and manage continuous queries on such paradigm should be studied in order to minimize the cost.

Acknowledgment

This thesis represents not only my research at Osaka University but also a part of my life that has a good opportunity to study and live in Japan these years. Completion of this thesis was possible because of the supports and efforts of a number of people.

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Takahiro Hara, who always gives me countless opportunities from the first step in this laboratory. It has been an honor to be his PhD student. He has been doing his best cultivating me to grow as a good researcher from my Master through PhD. This thesis would not have finished without his valuable advice and his guidance.

I am grateful to my thesis committee members, Professor Makoto Onizuka and Professor Yasuyuki Matsushita at the Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University for their valuable time giving useful comments towards improving and sharpening this thesis.

I am thankful to Professor Shojiro Nishio, President of Osaka University who was my advisor during my Master for being attentive and caring about my study for these years, Professor Tomoki Yoshihisa, Professor Masumi Shirakawa and Professor Takuya Maekawa for actively working for all students and always kindly giving the valuable guidance as well as many opportunities.

I would like to acknowledge my research team both past and present members, Professor Akimitsu Kanzaki, Professor Daichi Amagata, Dr. Yuya Sasaki, Dr. Keisuke Goto, Dr. Yuka Komai, Mr. Masahiro Yokoyama, Mr. Yuki Nakayama, Mr. Boqi Gao, Mr. Shuhei Hayashida and Mr. Syunya Nishio. Also, it is my pleasure to work with all smart members in Hara laboratory. All of them have given me cooperative and active supports throughout my study.

Last but not least, I would like to thank to my family who always respects my every

decision, stands by my side even good or bad times and wholeheartedly supports me on every step of my life.

At the end, I am thankful to all of my acquaintances who share positive vibes and always cheer me up. This means so much to me.

REFERENCE

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proc. of the European Conf. on Computer Systems (EuroSys)*, pages 29–42, 2013.

[2] C. C. Aggarwal and P. S. Yu. A survey of uncertain data algorithms and applications. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 21(5):609–623, 2009.

[3] A. Alamri, W. S. Ansari, M. M. Hassan, M. S. Hossain, A. Alelaiwi, and M. A. Hossain. A survey on sensor-cloud: architecture, applications, and approaches. *International Journal of Distributed Sensor Networks*, 2013.

[4] L.-M. Ang and K. P. Seng. Big sensor data applications in urban environments. *Big Data Research*, 2016.

[5] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 28–39, 2003.

[6] E. Baikousi and P. Vassiliadis. Maintenance of top-k materialized views. *Distributed and Parallel Databases*, 27(2):95–137, 2010.

[7] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[8] I. Bartolini, P. Ciaccia, and M. Patella. Efficient sort-based skyline evaluation. *ACM Trans. on Database Systems (TODS)*, 33(4):31, 2008.

[9] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

[10] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 421–430, 2001.

[11] K. C.-C. Chang and S.-w. Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 346–357, 2002.

[12] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(1):377–409, 1993.

[13] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang. A safe zone based approach for monitoring moving skyline queries. In *Proc. of Int'l Conf. on Extending Database Technology (EDBT)*, pages 275–286, 2013.

[14] M. A. Cheema, Z. Shen, X. Lin, and W. Zhang. A unified framework for efficiently processing ranking related queries. In *Proc. of Int'l Conf. on Extending Database Technology (EDBT)*, pages 427–438, 2014.

[15] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 16(9):1112–1127, 2004.

[16] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Indexing for vector projections. In *Proc. of Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 367–376, 2011.

[17] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Indexing reverse top-k queries in two dimensions. In *Proc. of Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 201–208, 2013.

[18] C. T. Chou, N. Bulusu, and S. Kanhere. Sensing data market. In *Proc. of Int'l Conf. on Distributed Computing in Sensor Systems (DCOSS)*, 2007.

[19] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 48–48, 2006.

[20] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis. Answering top-k queries using views. In *Proc. of Int'l Conf. on Very Large Data Bases (VLDB)*, pages 451–462, 2006.

[21] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *Proc. of Int'l Conf. on Very Large Data Bases (VLDB)*, pages 291–302, 2007.

[22] X. Ding, X. Lian, L. Chen, and H. Jin. Continuous monitoring of skylines over uncertain data streams. *Information Sciences*, 184(1):196 – 214, 2012.

[23] P. Dutta, P. M. Aoki, N. Kumar, A. Mainwaring, C. Myers, W. Willett, and A. Woodruff. Common sense: participatory urban sensing using a network of handheld air quality monitors. In *Proc. of Int'l Conf. on Embedded Networked Sensor Systems (SenSys)*, pages 349–350, 2009.

[24] T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Indexing uncertain spatio-temporal data. In *Proc. of Int'l Conf. on Information and Knowledge Management (CIKM)*, pages 395–404, 2012.

[25] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003.

[26] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *Proc. of Int'l Conf. on Very Large Data Bases (VLDB)*, pages 229–240, 2005.

[27] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB Journal*, 16(1):5–28, Jan. 2007.

[28] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[29] K. Hose and A. Vlachou. A survey of skyline processing in highly distributed environments. *VLDB Journal*, 21(3):359–384, 2012.

[30] V. Hristidis, N. Koudas, and Y. Papakonstantinou. Prefer: A system for the efficient execution of multi-parametric ranked queries. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 259–270, 2001.

[31] Y.-L. Hsueh, R. Zimmermann, and W.-S. Ku. Efficient updates for continuous skyline computations. In *Proc. of Int'l Conf. on Database and Expert Systems Applications (DEXA)*, pages 419–433, 2008.

[32] M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 673–686, 2008.

[33] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in manets. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 66–66, 2006.

[34] Z. Huang, H. Lu, B. C. Ooi, and A. Tung. Continuous skyline queries for moving objects. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 18(12):1645–1658, 2006.

[35] O. C. Ibe. *Elements of Random Walk and Diffusion Processes*. John Wiley & Sons, 2013.

[36] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[37] H. Jafarpour, B. Hore, S. Mehrotra, and N. Venkatasubramanian. Subscription subsumption evaluation for content-based publish/subscribe systems. In *Proc. of Int'l Conf. on Middleware (Middleware)*, pages 62–81, 2008.

[38] B. Jiang and J. Pei. Online interval skyline queries on time series. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 1036–1047, 2009.

[39] H. Jiang, J. Cheng, D. Wang, C. Wang, and G. Tan. Continuous multi-dimensional top-k query processing in sensor networks. In *Proc. of Int'l Conf. on Computer Communications (INFOCOM)*, pages 793–801, 2011.

[40] Y. Komai, Y. Sasaki, T. Hara, and S. Nishio. Nn query processing methods in mobile ad hoc networks. *IEEE Trans. on Mobile Computing*, 13(5):1090–1103, 2014.

[41] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *ACM Journal*, 22(4):469–476, Oct. 1975.

[42] J. Lee, H. Cho, and S.-w. Hwang. Efficient dual-resolution layer indexing for top-k queries. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 1084–1095, 2012.

[43] J.-S. Lee and B. Hoh. Sell your experiences: a market mechanism based incentive for participatory sensing. In *Proc. of Int'l Conf. on Pervasive Computing and Communications (PerCom)*, pages 60–68, 2010.

[44] M.-W. Lee and S.-W. Hwang. Continuous skylining on volatile moving data. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 1568–1575, 2009.

[45] Y. W. Lee, K. Y. Lee, and M. H. Kim. Efficient processing of multiple continuous skyline queries over a data stream. *Information Sciences*, 221:316–337, 2013.

[46] H. Lu, Y. Zhou, and J. Haustad. Efficient and scalable continuous skyline monitoring in two-tier streaming settings. *Information Systems*, 38(1):68–81, 2013.

[47] T. H. Luan, L. Gao, Z. Li, Y. Xiang, and L. Sun. Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815*, 2015.

[48] S. Madria, V. Kumar, and R. Dalvi. Sensor cloud: A cloud of virtual sensors. *IEEE Software*, 31(2):70–77, 2014.

[49] M. Morse, J. M. Patel, and W. I. Grosky. Efficient continuous skyline computation. *Information Sciences*, 177(17):3411–3437, 2007.

[50] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 635–646, 2006.

[51] M. J. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT press, 1994.

[52] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. on Database Systems (TODS)*, 30(1):41–82, 2005.

[53] O. Papapetrou and M. Garofalakis. Continuous fragmented skylines over distributed streams. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 124–135, 2014.

[54] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proc. of Int'l Conf. on Very Large Data Bases (VLDB)*, pages 15–26, 2007.

[55] C. Peng, T. G. Kolda, and A. Pinar. Accelerating community detection by using k-core subgraphs. *CoRR*, abs/1403.2226, 2014.

[56] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Sensing as a service model for smart cities supported by internet of things. *Trans. on Emerging Telecommunications Technologies*, 25(1):81–93, 2014.

[57] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Proc. of Int'l Symposium on Advances in Spatial Databases (SSD)*, pages 111–131, 1999.

[58] 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.

[59] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 886–895, 2007.

[60] C. J. V. Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979.

[61] N. H. Ryeng, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient distributed top-k query processing with caching. In *Proc. of Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 280–295, 2011.

[62] G. Sagy, D. Keren, I. Sharfman, and A. Schuster. Distributed threshold querying of general functions by a difference of monotonic representation. In *Proc. of Int'l Conf. on Very Large Data Bases (VLDB)*, volume 4, pages 46–57, 2010.

[63] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and Ü. V. Çatalyürek. Streaming algorithms for k-core decomposition. In *Proc. of Int'l Conf. on Very Large Data Bases (VLDB)*, volume 6, pages 433–444, 2013.

[64] Y. Sasaki, R. Hagihara, T. Hara, M. Shinohara, and S. Nishio. A top-k query method by estimating score distribution in mobile ad hoc networks. In *Proc. of Int'l Conf. on Advanced Information Networking and Applications Workshops (WAINA)*, pages 944–949, 2010.

[65] A. Shastri, Y. Di, E. A. Rundensteiner, and M. O. Ward. MTopS: scalable processing of continuous top-k multi-query workloads. In *Proc. of Int'l Conf. on Information and Knowledge Management (CIKM)*, pages 1107–1116, 2011.

[66] X. Sheng, J. Tang, X. Xiao, and G. Xue. Sensing as a service: Challenges, solutions and future directions. *Sensors Journal*, 13(10):3733–3741, 2013.

[67] A. S. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 68–68, 2006.

[68] M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Probabilistic top-k and ranking-aggregate queries. *ACM Trans. on Database Systems (TODS)*, 33(3):13, 2008.

[69] M. A. Soliman, I. F. Ilyas, and K. Chen-Chuan Chang. Top-k query processing in uncertain databases. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 896–905, 2007.

[70] A. Sultana, N. Hassan, C. Li, J. Yang, and C. Yu. Incremental discovery of prominent situational facts. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 112–123, 2014.

[71] S. Sun, Z. Huang, H. Zhong, D. Dai, H. Liu, and J. Li. Efficient monitoring of skyline queries over distributed data streams. *Knowledge and Information systems*, 25(3):575–606, 2010.

[72] E. W. Swokowski. *Calculus with Analytic Geometry*. Taylor & Francis, 1979.

[73] Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou. Branch-and-bound processing of ranked queries. *Information Systems*, 32(3):424 – 445, 2007.

[74] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 65–65, 2006.

[75] L. Tian, L. Wang, A. Li, P. Zou, and Y. Jia. Continuous skyline tracking on update data streams. In *Proc. of Int'l Workshops on Advances in Web and Network Technologies, and Information Management (APWeb/WAIM)*, pages 192–197, 2007.

[76] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *Proc. of Int'l Conf. on Distributed Computing Systems (ICDCS)*, pages 562–571, 2004.

[77] K. Udomlamlert, C. K. Adiputra, and T. Hara. Monitoring top-k on real-time dynamic social-network graphs. In *Proc. of Int'l Conf. on Distributed and Event-based Systems (DEBS)*, pages 317–321. ACM, 2016.

[78] K. Udomlamlert and T. Hara. Reducing expenses of top-k monitoring in sensor cloud services. In *Proc. of Int'l Conf. on Distributed and Event-based Systems (DEBS)*, pages 187–198, 2016.

[79] K. Udomlamlert, T. Hara, and S. N. D. Threshold-based distributed continuous top-k query processing for minimizing communication overhead. *IEICE Trans. on Information and Systems*, E99-D(2):383–396, 2 2016.

[80] K. Udomlamlert, T. Hara, and S. Nishio. Continuous top-k query processing on horizontally-distributed data. In *Proc. of Domestic Conf. on FIT*, 2013.

[81] K. Udomlamlert, T. Hara, and S. Nishio. Communication-efficient preference top-k monitoring queries via subscriptions. In *Proc. of Int'l Conf. on Scientific and Statistical Database Management (SSDBM)*, pages 44:1–44:4, 2014.

[82] K. Udomlamlert, T. Hara, and S. Nishio. Skyline calculation on frequent data updates (in japanese). In *Proc. of Domestic Conf. on DPSWS*, pages 252–260, 2014.

[83] K. Udomlamlert, T. Hara, and S. Nishio. Candidate pruning technique for skyline computation over frequent update streams. In *Proc. of Int'l Conf. on Database and Expert Systems Applications (DEXA)*, pages 93–108, 2015.

[84] K. Udomlamlert, T. Hara, and S. Nishio. Reducing expenses of sensor-cloud services for dynamic skyline monitoring. In *Proc. of Domestic Conf. on DEIM*, 2016.

[85] K. Udomlamlert, T. Hara, and S. Nishio. Subscription-based data aggregation techniques for top-k monitoring queries. *World Wide Web*, (to appear).

[86] A. Vlachou, C. Doulkeridis, and K. Nørvåg. Distributed top-k query processing by exploiting skyline summaries. *Distributed and Parallel Databases*, 30(3-4):239–271, 2012.

[87] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. On efficient top-k query processing in highly distributed environments. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 753–764, 2008.

[88] H. Wang, Y. Cai, Y. Yang, N. Mamoulis, et al. Durable queries over historical time series data. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 26(3):595–607, 2014.

[89] X. Wang, A. V. Vasilakos, M. Chen, Y. Liu, and T. T. Kwon. A survey of green mobile networks: Opportunities and challenges. *Mobile Networks and Applications*, 17(1):4–20, 2012.

[90] Y. Wang, G. Cong, G. Song, and K. Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *Proc. of Int'l Conf. on Knowledge Discovery and Data Mining (SIGKDD)*, pages 1039–1048, 2010.

[91] M. Wu, J. Xu, X. Tang, and W.-C. Lee. Top-k monitoring in wireless sensor networks. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 19(7):962–976, 2007.

[92] M. Xie, L. V. Lakshmanan, and P. T. Wood. Efficient top-k query answering using cached views. In *Proc. of Int'l Conf. on Extending Database Technology (EDBT)*, pages 489–500, 2013.

[93] J. Xin, G. Wang, L. Chen, et al. Continuously maintaining sliding window skylines in a sensor network. In *Proc. of Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 509–521, 2007.

[94] D. Yang, A. Shastri, E. A. Rundensteiner, and M. O. Ward. An optimal strategy for monitoring top-k queries in streaming windows. In *Proc. of Int'l Conf. on Extending Database Technology (EDBT)*, pages 57–68, 2011.

[95] M. L. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *Proc. of Int'l Conf. on Very Large Data Bases (VLDB)*, pages 483–494, 2007.

[96] A. Yu, P. K. Agarwal, and J. Yang. Processing a large number of continuous preference top-k queries. In *Proc. of Int'l Conf. on Management of Data (SIGMOD)*, pages 397–408, 2012.

[97] A. Yu, P. K. Agarwal, and J. Yang. Processing and notifying range top-k subscriptions. In *Proc. of Int'l Conf. on Data Engineering (ICDE)*, pages 810–821, 2012.

[98] Y. Zhang, R. Cheng, and J. Chen. Evaluating continuous probabilistic queries over imprecise sensor data. In *Proc. of Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 535–549, 2010.

[99] Y. Zhang, J. Zhou, and J. Cheng. Preference-based top-k influential nodes mining in social networks. In *Proc. of Int'l Conf. on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1512–1518. IEEE, 2011.

[100] K. Zhao, Y. Tao, and S. Zhou. Efficient top-k processing in large-scaled distributed environments. *Data and Knowledge Engineering*, 63(2):315 – 335, 2007.

[101] L. Zou and L. Chen. Pareto-based dominant graph: An efficient indexing structure to answer top-k queries. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 23(5):727–741, 2011.