Minimizing the Communication Cost for Continuous Skyline Maintenance

Zhenjie Zhang[1], Reynold Cheng[2], Dimitris Papadias[3], Anthony K.H. Tung[1]
[1]School of Computing
National University of Singapore
{zhenjie,atung}@comp.nus.edu.sg

[2]Department of Computer Science
Hong Kong University
ckcheng@cs.hku.hk

[3]Department of Computer Science & Engineering
Hong Kong University of Science and Technology
dimitris@cse.ust.hk

ABSTRACT

Existing work in the skyline literature focuses on optimizing the processing cost. This paper aims at minimization of the communication overhead in client-server architectures, where a server continuously maintains the skyline of dynamic objects. Our first contribution is a *Filter* method that avoids transmission of updates from objects that cannot influence the skyline. Specifically, each object is assigned a filter so that it needs to issue an update only if it violates its filter. *Filter* achieves significant savings over the naive approach of transmitting all updates. Going one step further, we introduce the concept of *frequent skyline query over a sliding window* (FSQW). The motivation is that snapshot skylines are not very useful in streaming environments because they keep changing over time. Instead, FSQW reports the objects that appear in the skylines of at least $\theta \cdot s$ of the s most recent timestamps ($0 < \theta \leq 1$). *Filter* can be easily adapted to FSQW processing, however, with potentially high overhead for large and frequently updated datasets. To further reduce the communication cost, we propose a *Sampling* method, which returns approximate FSQW results without computing each snapshot skyline. Finally, we integrate *Filter* and *Sampling* in a *Hybrid* approach that combines their individual advantages.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications

General Terms

Algorithm, Performance

Keywords

Skyline Query, Continuous Query, Communication

1. INTRODUCTION

Skyline computation has received considerable attention in both conventional databases and stream environments. While the existing approaches focus exclusively on the minimization of the processing cost, in many applications the real bottleneck is the network overhead due to update transmissions. Assume, for instance, a server that receives readings (e.g., temperature, humidity, pollution level) from various sensors and continuously maintains the skyline of these readings in order to identify potentially problematic situations (e.g., the most extreme combinations of values). The server has substantial resources so that optimization of its computational overhead is not critical. On the other hand, the sensor devices are usually battery-powered and should conserve energy. Usually, uplink messages, sent to the server for updates, constitute the most important factor for energy consumption [8] and should be minimized. As another example, consider a system that monitors network traffic such as Cisco NetFlow. The server usually collects detailed traffic logs on a per flow granularity, which account for hundreds of GBytes of data per day. Skyline monitoring can be used to detect potential traffic congestions, or attacks on the network. In this case, minimization of update frequency is important for reducing the amount of network traffic to the server.

Our setting is a client-server architecture, where the server receives records from various sources/clients[1]. A record r_i has d ($d > 1$) attributes, each taking values from a totally ordered domain. Therefore, it can be represented as a point p_i in d-dimensional space, and in the sequel we use the terms record/tuple/point/object interchangeably. The server receives updates from the sources on their corresponding records at discrete timestamps. An update alters the value of at least one attribute, and it corresponds to a movement of the respective point to a new position. Records can be inserted and deleted at any timestamp. Insertions and deletions can be thought of as movements from/to a non-existent position. We use p_i^t to denote the status of point p_i (record r_i) at time t: $p_i^t = (p_i^t[1], p_i^t[2], \ldots, p_i^t[d])$, for each $1 \leq k \leq d$. A snapshot $S_t = \{p_1^t, p_2^t, \ldots, p_n^t\}$ at timestamp t contains all points alive at t, i.e., all records that have been

[1]For simplicity, we assume that each record originates from a unique source, but the proposed techniques are applicable if the same client transmits multiple records.

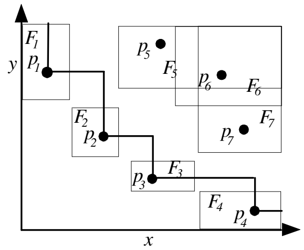


Figure 1: Example of skyline filters

inserted before t, but have not been deleted at t. We say that p_i^t *dominates* p_j^t, if $p_i^t[k]$ is at least as good as $p_j^t[k]$ for all k, and there is an attribute l such that $p_i^t[l]$ is better than $p_j^t[l]$. The skyline $Sky(S_t)$ of a snapshot S_t is the subset of the records not dominated by any other point in S_t.

Our first contribution is a *Filter* method that continuously maintains the skyline, while avoiding transmission of updates from objects that cannot influence it. Specifically, the server computes, for each record, a hyper-rectangle that bounds the value of every attribute. These rectangles are transmitted through downlink messages to the corresponding clients (e.g., sensors). A client needs to issue an update only if the point has moved out of its filter (i.e., some attribute has exceeded the range imposed by the server), or if the server explicitly asks for the current attribute values. Figure 1 illustrates a snapshot skyline (p_1 to p_4) over seven records, assuming that lower values are preferable on each axis. Every point p_i is associated with a filter F_i. In order for the skyline to change at some subsequent timestamp, at least one point must exit its current filter; as long as the records remain within their respective filters, all location updates can be avoided.

Filter can capture the exact skyline at each timestamp, and in most settings achieves significant savings in terms of network overhead. However, in several applications, snapshot skylines may not be important because they change too fast to be meaningful. Furthermore, in the presence of communication errors and outliers in the data, it is more interesting to identify the records that consistently appear in the skyline over several timestamps. Motivated by this observation, we introduce the concept of *frequent skyline query over a sliding window* (FSQW). A window W_t^s is as a set of s consecutive snapshots ending at t, i.e. $W_t^s = \{S_{t+1-s}, \ldots, S_t\}$. A record constitutes a θ-frequent skyline point in W_t^s if it appears in at least $\theta \cdot s$ snapshot skylines within the window ($0 < \theta \leq 1$). A FSQW continuously reports the frequent skyline points as the sliding window moves along the time dimension.

Filter can be trivially adapted for exact processing of FSQW. However, despite its savings with respect to the naive method of transmitting all updates, it may still require a large number of update messages. To alleviate this problem, we propose a *Sampling* method, in which updates are transmitted at certain instances, depending on the desired trade-off between accuracy and message overhead. Finally, we integrate *Filter* and *Sampling* in a *Hybrid* approach, which differentiates three modes for each record: *filter*, *sampling*, or *mixed* mode. *Hybrid* has a balanced behavior even under extreme settings, where the performance of *Filter* and *Sampling* deteriorates.

The rest of the paper is organized as follows. Section 2 reviews related work on skylines and minimization of the network overhead in other query types. Section 3 presents the necessary definitions and discusses some interesting skyline properties in our setting. Section 4 introduces *Filter* and analyzes its performance. Section 5 presents *Sampling* and provides guidelines for setting the sampling rate. Section 6 describes *Hybrid* and switching to different modes. Section 7 evaluates our methods through extensive experiments, and Section 8 concludes the paper.

2. RELATED WORK

The skyline operator was first introduced to the database community in [3]. Since then a large number of algorithms have been proposed for conventional databases. These methods can be classified in two general categories depending on whether they use indexes (e.g., *Index* [26], *Nearest Neighbor* [15], *Branch and Bound Skyline* [22]) or not (e.g., , *Divide and Conquer*, *Block Nested Loop*[3], *Sort First Skyline*[7, 10]). Furthermore, skylines have been studied in the context of mobile devices [13], distributed systems [2], and unstructured [30] as well as structured networks [28].

In addition, several papers focus on skyline computation when the dataset has some specific properties. [4] extends *Branch and Bound Skyline* for the case where some attributes take values from partially-ordered domains. [19] focuses on skyline processing for domains with low cardinality. [5] deals with high dimensional skylines. Finally, a number of interesting variants of the basic definition have been proposed. *Skyline cubes* [32, 31] compute the skylines in a subset or all subspaces. *Probabilistic skylines* [23] assume that each record has several instances, in which case the dominance relationship is probabilistic. *Spatial skylines* [25] return the set of data points that can be the nearest neighbors of any point in a given query set. A *reverse skyline* [9] outputs the records whose dynamic skyline contains a query point.

The above methods deal with snapshot query processing and do not include mechanisms for maintaining the skyline in the presence of updates. On the other hand, [29] proposes a space decomposition for re-computing the skyline when a record is deleted. [16] applies z-ordering to achieve efficient insertion as well as deletion. [17] and [27] study skyline maintenance over sliding windows, utilizing some interesting properties to expunge records (before their expiration) that cannot become part of the skyline. Morse et al. [18] assume streams, where the records are explicitly deleted or modified independently of their arrival order. In all cases, the assumption is that the server receives all updates, and the goal is to minimize its processing cost. Thus, these techniques are orthogonal to ours in the sense that they can be integrated within a system that optimizes both the processing and the transmission cost during skyline maintenance.

Although minimization of the communication overhead in client-server architectures is new to the skyline literature, it has been applied before to monitoring of spatial queries. *Q-index* [24] assumes a central server that receives the positions of objects, while maintaining the results of continuous range queries. In order to reduce the number of location updates, the server transmits to each object a rectangular or circular *safe region*, such that the object does not need to issue an update as long as it remains within its region. Figure 2 illustrates the safe regions of two points p_1 and p_2, given six running range queries q_1 to q_6. While p_2 is in its safe

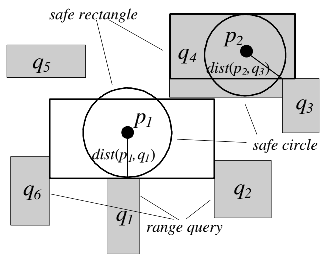


Figure 2: Example of safe regions

rectangle or circle, it belongs to the result of q_4. As soon at it exits the safe region, it may stop being in the result of q_4 and/or start being in the range of q_3. Similarly, p_1 cannot influence any query while it remains within its safe region. Analogous concepts have also been applied to continuous nearest neighbors in [12, 20].

In Data Stream Management Systems (DSMS), *stream filters* have been used to offload some processing from the server [21, 14, 6]. In particular, each stream source is installed with a simple filter, so that a data item is sent to the central server only if its value satisfies the conditions defined in the filters. For instance, Babcock and Olston [1] consider a scenario where a central server continuously reports the largest k values obtained from distributed data streams. Their method maintains arithmetic constraints at the sources to ensure that the most recently reported answers remain valid. Up-to-date information is obtained only when some constraint is violated, thus reducing the communication overhead.

These concepts are similar in principle to the proposed *Filter* method, with however an important difference. For spatial ranges, a safe region is based on the object's location with respect to each query range, independently of the other objects in the dataset. For nearest neighbors, safe region computation takes into account just a few objects around the queries (typically, only the NNs). Similarly, the filters used in DSMS can be easily computed using the conditions imposed by the query. On the other hand, for the skyline there are no queries; instead, the filter of a record depends on the attribute values (or the filters) of numerous other tuples. Therefore, as we show in the subsequent sections, filter computation in our context is more complex and expensive.

3. DEFINITIONS AND PRELIMINARIES

We assume a time-slotted system, where each client notifies the server about updates at discrete timestamps, i.e., there is a minimum interval dt between two consecutive updates of the same record, such that the round-trip time of a message between the server and any client is negligible compared to dt. An *uplink* message refers to a transmission from a client to the server, and a *downlink* message to the opposite direction. If c_u, N_u (resp. c_d, N_d) is the cost and cardinality of uplink (resp. downlink) messages, the total transmission overhead of the system can be measured as $c_uN_u + c_dN_d$. The problem we intend to solve in this paper is to minimize this cost, while maintaining the exact or approximate skyline over time.

The status of record r_i at time t corresponds to a point in the d-dimensional unit space $p_i^t = (p_i^t[1], p_i^t[2], \ldots, p_i^t[d])$, where $0 \le p_i^t[k] \le 1$ for each $1 \le k \le d$. Records can be updated or deleted at any timestamp after their insertion (i.e., there is no particular order depending on their arrival, as in the sliding window model). A snapshot S_t at time t contains all records alive at time t. A window W_t^s contains s consecutive snapshots ending at S_t, i.e., $W_t^s = \{S_{t+1-s}, \ldots, S_t\}$. To simplify notation, we omit the timestamp, when it is clear from the context or not important for the discussion. Without loss of generality, in order to determine dominance relationships, we assume that smaller attribute values are preferable over larger ones.

Definition 1. ***Point Dominance***
A point p_i dominates another p_j at time t, if $p_i^t[k] \le p_j^t[k]$ for all k, and $p_i^t[l] < p_j^t[l]$ for at least one attribute l.

A *filter* F_i^t is a hyper-rectangle that covers point p_i^t at time t. F_i^t is defined by d pairs of boundaries ($F_i^t.l[1]$, $F_i^t.u[1]$), $\ldots$, ($F_i^t.l[d]$, $F_i^t.u[d]$), where each pair ($F_i^t.l[k]$, $F_i^t.u[k]$) is the lower and the upper bound of the filter on dimension k. $F_i^t.l$ and $F_i^t.u$ denote the lower-left and upper-right corner of the filter, respectively. Since every point p_i is associated with at most one filter at any timestamp, we misuse F_i as replacement of F_i^t when no ambiguity occurs. Intuitively, a filter constrains a point whose exact location is unknown. A client needs to issue an update to the central server in two different situations: *filter failure* and *probe request*. A failure occurs when a record r_i moves out of its filter F_i; otherwise, we say that F_i is *valid*. A probe request happens when the central server asks for the exact value of a record. For instance, in Figure 1, if p_1 causes a filter failure, the corresponding client has to issue an update. Upon receiving the update, the server may probe for the current status of other records (e.g., p_2) in order to determine if there is a change in the dominance relationships. Note that in the example of Figure 2 the violation of a spatial filter does not involve any probe because a range filter is computed solely on the position of the object with respect to the queries. Next, we generalize the definition of dominance to capture the case where a record r_i is represented either by a point p_i, or a filter F_i.

Definition 2. ***Certain Dominance***
A record r_i certainly dominates another r_j at timestamp t, if at least one of the following conditions holds: (1) p_i^t dominates p_j^t; (2) $F_i^t.u$ dominates p_j^t; (3) p_i^t dominates $F_j^t.l$; (4) $F_i^t.u$ dominates $F_j^t.l$, where point dominance is based on Definition 1.

Definition 3. ***Possible Dominance***
A record r_i possibly dominates another r_j at timestamp t, if at least one of the following conditions holds: (1) $F_i^t.l$ dominates p_j^t; (2) p_i^t dominates $F_j^t.u$; (3) $F_i^t.l$ dominates $F_j^t.u$, where point dominance is based on Definition 1.

Clearly, if r_i certainly dominates r_j, it also possibly dominates r_j, but the opposite is not true. The concept of possible dominance is only applicable when at least one of the two records is represented by a filter and certain dominance cannot be established. When there is no ambiguity, we use the term dominance to also refer to certain dominance. In Figure 1, $F_2.u$ dominates $F_6.l$; consequently r_2 dominates r_6, even if the exact attribute values of both records are unknown (provided that their filters are valid). On the other hand, assuming that r_5 and r_6 are represented by F_5 and F_6, they both possibly dominate each other (note that $F_5.l$ dominates $F_6.u$ and $F_6.l$ dominates $F_5.u$).

DEFINITION 4. ***Snapshot Skyline*** $Sky(S_t)$
A skyline $Sky(S_t)$ over snapshot S_t is the set of all alive records that are not dominated at timestamp t.

DEFINITION 5. ***Frequent Skyline Point***
A record r_i is a θ-frequent skyline point in the window W_t^s, if r_i appears in at least $\theta \cdot s$ skylines within W_t^s.

DEFINITION 6. ***Frequent Skyline Query over Sliding Window*** $FSQW(\theta, W_t^s)$
Given a threshold θ ($0 < \theta \leq 1$), $FSQW(\theta, W_t^s)$ returns the set of all θ-frequent skyline points over W_t^s.

Note that the snapshot skyline constitutes a special case of FSQW, where both s and θ equal 1. Figure 3 includes three consecutive snapshots over 7 records. At S_t, the skyline is $Sky(S_t) = \{p_1, p_2, p_3, p_4\}$. At S_{t+1}, p_7 replaces p_1 in $Sky(S_{t+1})$. At S_{t+2}, $Sky(S_{t+2}) = \{p_2, p_4, p_7\}$. Assuming $\theta = 0.5$ and considering the window W_{t+2}^3, $FSQW(0.5, W_{t+2}^3) = \{p_2, p_3, p_4, p_7\}$. If the threshold θ is raised to 0.7, the result contains only two records, $\{p_2, p_4\}$. Given the skyline at each snapshot in W_t^s, the server can calculate $FSQW(\theta, W_t^s)$ by simply counting the frequency of each record. A more interesting question is whether the server can obtain the exact $FSQW$ results without computing the skyline at each timestamp.

THEOREM 3.1. *Every algorithm that returns exact $FSQW$ results must compute the exact snapshot skyline at each timestamp.*

PROOF. Let $\mathbb{A}$ be an algorithm for FSQW processing that does not compute the skyline for some timestamp t. We can always construct a data set that leads $\mathbb{A}$ to erroneous results as follows.

If $\mathbb{A}$ misses a skyline point p_i in $Sky(S_t)$, we generate a data set with p_i in exactly $\theta s - 1$ skylines at the following $s-1$ timestamps. At time $t+s-1$, algorithm $\mathbb{A}$ will omit p_i from $FSQW(\theta, W_{t+s-1}^s)$, although it should be reported.

If algorithm $\mathbb{A}$ wrongly includes p_i in $Sky(S_t)$, we also construct a data set with p_i in exactly $\theta s - 1$ skylines at the following $s-1$ timestamps. At time $t+s-1$, $\mathbb{A}$ will report p_i in $FSQW(\theta, W_{t+s-1}^s)$, although it should be excluded. $\square$

The implication of the theorem is that the skyline computation at each timestamp is unavoidable for any exact FSQW algorithm. In the next section, we utilize filters to reduce the communication cost. The proposed method is applicable to both snapshot skylines and, consequently FSQW processing.

4. FILTER METHOD

Section 4.1 introduces the general algorithmic framework of *Filter*. Section 4.2 proposes a model for the update cost, and utilizes this model to provide algorithms for filter generation.

4.1 Filter Framework

Filter follows the framework summarized in Algorithm 1. For generality, we present the version for FSQW processing since it subsumes snapshot skyline computation. The server first receives the initial status of all objects, generates the skyline, computes the filters, and transmits them to the clients. At every subsequent timestamp t, it re-computes the current skyline and the result of each $FSQW(\theta, W_t^s)$ installed in the system[2]. Finally, the server updates the filters of the objects (if necessary), and sends them to the affected clients. Following the literature of DSMS, we assume that processing takes place entirely in main memory. In the following, we cover the details of **ComputeSkyline** and **FilterConstruction** invoked in Algorithm 1.

Algorithm 1 Filter (window size s, threshold θ)

```
Get the initial status of the records {r_1^0, ..., r_n^0}
Sky(S_0) = ComputeSkyline(S_0)
FilterConstruction(S_0)
Send filters to the clients
for each timestamp t do
    Receive updates due to filter violations
    Sky(S_t) = ComputeSkyline(S_t)
    for each r_i in S_t and each FSQW(θ, W_t^s) do
        if r_i is a θ-frequent skyline point in W_t^s then
            Output r_i as part of FSQW result
    FilterConstruction(S_t)
    Send updates to objects with new filter
```

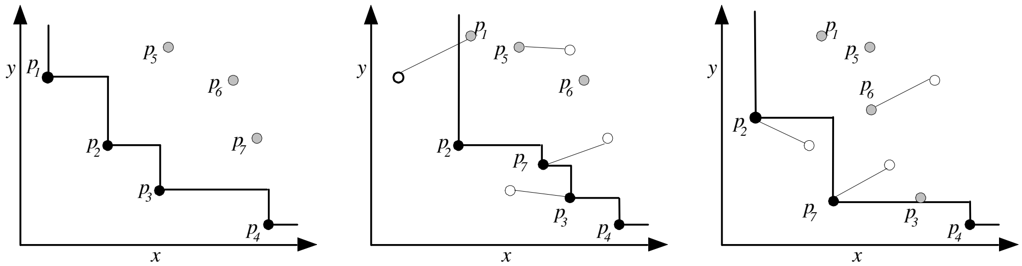
Algorithm 2 illustrates skyline computation on the current snapshot S_t. If a record r_i is certainly dominated by another r_j according to Definition 2, it is discarded immediately. If r_j possibly dominates r_i by Definition 3, r_j is inserted into a candidate dominator list H. After this round, if H is not empty, we need to continue in order to determine whether r_i is in skyline. If the server has not received an update for r_i at the current timestamp (because F_i is still valid), it sends a probe request to obtain its current status p_i^t. If after the probe, any object in H dominates p_i^t, r_i is discarded. Otherwise, the server probes the up-to-date versions for all records in the candidate list. If no point dominates p_i^t, p_i^t is inserted into the skyline.

Note that Algorithm 2 minimizes the number of probes, by first resolving dominance relationships that do not require any probes. Then, it obtains the current status of r_i with a single probe. Only if r_i remains a skyline candidate after the above tests, the server probes records in H. Given a set of properly generated filters, it is easy to verify the correctness of the algorithm since each record is compared against every other record, unless it is dominated. All ambiguities regarding dominance relationships are resolved by probes. Therefore, the skyline contains all non-dominated records and no false hits.

4.2 Filter Construction

In Algorithm 1, the function **FilterConstruction** generates a filter F_i for each record r_i without a valid filter. Before proceeding to its description, we study some filter requirements. Recall that filter failures trigger updates, which in turn determine the skyline. To detect all skyline updates, the filters should be constructed in some way that guarantees that as long as there is no failure, there cannot be any changes in the skyline. The following lemmas, following

[2]Depending on the application, there may be multiple FSQW with different threshold θ and window size s parameters.



(a) At time t (b) At time $t+1$ (c) At time $t+2$

Figure 3: Examples of snapshot skylines and FSQW

Algorithm 2 ComputeSkyline (snapshot S_t)

1: Construct skyline buffer S
2: Construct a candidate dominator list H
3: **for** each record r_i **do**
4:   Clear H
5:   **for** each record r_j $(j \neq i)$ **do**
6:     **if** r_j certainly dominates r_i **then**
7:       Discard r_i and go to (3)
8:     **if** r_j possibly dominates r_i **then**
9:       Append r_j to H
10:   **if** H is not empty **then**
11:     **if** filter F_i is still valid **then**
12:       $p_i^t =$**Probe**(r_i)
13:       **if** any object in H certainly dominates p_i^t **then**
14:         Discard r_i and go to (3)
15:     **for** each $r_j \in H$ with valid filter F_j **do**
16:       $p_j^t =$**Probe**(r_j)
17:       **if** p_j^t dominates p_i^t **then**
18:         Discard r_i and go to (3)
19:   Append r_i into skyline
20: Return S

---

from Definitions 2 and 3, provide sufficient conditions for the above requirements.

Lemma 4.1. *A record r_i is in the skyline, if no filter F_j $(j \neq i)$ possibly dominates F_i.*

Lemma 4.2. *A record r_i is not in the skyline, if there is at least one filter F_j $(j \neq i)$ that certainly dominates F_i.*

If F_j certainly dominates F_i, we say that (F_j, F_i) is a *filter dominance pair*, or F_j is the *dominator* of F_i. A filter set $\{F_1, F_2, \ldots, F_n\}$ is robust, if each skyline record satisfies Lemma 4.1 and each non-skyline record satisfies Lemma 4.2. The filter set of Figure 1 is robust because (i) none of F_1, F_2, F_3, F_4 is possibly dominated and (ii) all non-skyline filters are certainly dominated by a skyline filter (F_2 is the dominator of F_5, F_6, and F_3 is the dominator for F_7). Next, we qualitatively analyze the update cost of a filter-based method through the following theorem.

Theorem 4.1. *Any method following the framework of Algorithm 1 incurs update cost $c_u(X+Y) + c_d(X+2Y)$, where X is the number of filter failures, Y the number of probe requests and c_u (resp. c_d) is the cost of an uplink (resp. downlink) message.*

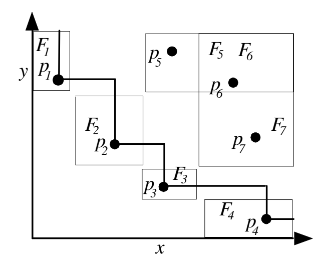


Figure 4: Alternative filter set

Proof. For each filter failure, the server receives one uplink message from a client and responds with a downlink message for a filter update. For each probe, the server sends a request, receives a response, and transmits back a new filter (i.e., two downlink and one uplink messages). Therefore, if there are X filter failures and Y probe requests, the update cost is at least $c_u(X+Y) + c_d(X+2Y)$. Since there is no additional communication overhead in the system, this is also the total cost. □

According to the previous theorem, in order to minimize the update cost, we have to reduce the number of filter failures and probe requests. However, these tasks are contradictory and difficult to optimize. For instance, Figure 4 illustrates an alternative filter set for the records of Figure 1, where the size of F_2, F_4 has increased, while that of F_1, F_3 has decreased. The enlargement of F_2 and F_4 delays their violations, but the reduction of F_1 and F_3 may cause their earlier failures, leading to probes on r_2 and r_4 for resolving uncertain dominance. A good filter should balance the probabilities of failure and probe requests. If $P_f(F_i)$ is the probability of filter failure and $P_r(F_i)$ is the probability of probe request on F_i, the expected update cost of F_i is $C(F_i) = c_u(P_f(F_i) + P_r(F_i)) + c_d(P_f(F_i) + 2P_r(F_i))$. The optimal filter set $\{F_1, \ldots, F_n\}$ should minimize

$$\sum_i C(F_i) = \sum_i ((c_u + c_d)P_f(F_i) + (c_u + 2c_d)P_r(F_i))$$

The next step concerns the derivation of $P_f(F_i)$ and $P_r(F_i)$. The probability of filter failure $P_f(F_i)$ can be estimated based on the shortest time that a violation can occur in F_i. Given a record r_i with filter F_i and maximum rate of

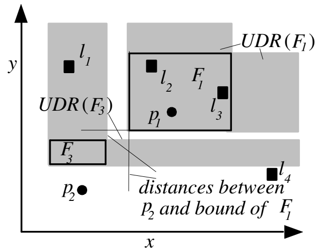


Figure 5: Examples of probe requests

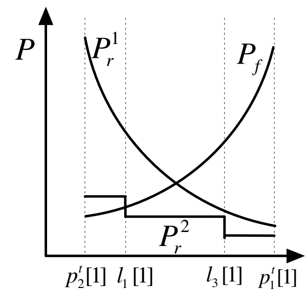


Figure 6: Probability functions on boundary value

change[3] C_i, the shortest time that r_i can reach the lower bound on dimension k is $(r_i[k] - F_i.l[k])/C_i$. Similarly, the shortest time to reach the upper bound on dimension k is $(F_i.u[k] - r_i[k])/C_i$. Since the failure happens only when the point hits one of the boundaries, the average probability of a failure on F_i is upper bounded by

$$P_f(F_i) = \frac{1}{2d} \sum_k \left(\frac{C_i}{F_i.u[k] - r_i[k]} + \frac{C_i}{r_i[k] - F_i.l[k]} \right)$$

As for the probability of a probe request, we note that there are two types of probes. The first happens when the previous dominator r_j ceases to dominate a non-skyline record r_i. The server needs the current status of r_i in order to determine if it becomes part of the skyline. Similar to the case of filter failure, we can estimate the probability using the distances from the dominator r_j to the lower bounds of F_i on all dimensions. Figure 5 shows the distances between the current location of dominator p_2 and the lower bounds of F_1. Based on these distances, the average probability for the first type of probes on a filter F_i can be estimated as

$$P_r^1(F_i) = \frac{1}{d} \sum_k \left(\frac{C_j}{F_i.l[k] - r_j[k]} \right)$$

The second type of probe request occurs to F_i when r_i possibly dominates another record r_j, in which case the server needs to determine whether r_j is in skyline. The status of r_j is recorded as a *probe source*. The server stores the most recent M probe sources in an array $L = \{l_1, l_2, \ldots, l_M\}$ to approximate their distribution. Given a filter F_i, the probability of second type probe requests on F_i is estimated by the ratio of recorded probes covered by $UDR(F_i)$ over their total number. $UDR(F_i)$ is the area dominated by $F_i.l$, but not dominated by $F_i.u$. Any point in $UDR(F_i)$ is possibly, but not certainly, dominated by F_i. We have

$$P_r^2(F_i) = \frac{|\{l_i \in L \mid l_i \in UDR(F_i)\}|}{M}$$

In Figure 5, the probe source log contains four locations, i.e. $L = \{l_1, l_2, l_3, l_4\}$. Two out of the four probe sources, l_2 and l_3, are in $UDR(F_1)$, while only l_1 is in $UDR(F_3)$. According to the previous definition, $P_r^2(F_1) = 1/2$ and $P_r^2(F_3) = 1/4$. Intuitively, $P_r^2(F_i)$ uses the past L probes to estimate the likelihood that a further probe will come from some point within $UDR(F_i)$, invalidating F_i. The first type of probes only happens to non-skyline records, while the second one can occur to all records. Therefore, the overall probability of a probe request can be summarized as:

$$P_r(F_i) = \begin{cases} P_r^2(F_i) & , \quad \text{if } p_i^t \in Sky(S_t) \\ P_r^1(F_i) + P_r^2(F_i), & \text{if } p_i^t \notin Sky(S_t) \end{cases}$$

Based on the above model, we first study the boundary optimization problem between two points on a single dimension. Specifically, we aim at the best lower (or upper) bound x on dimension k for filter F_1 with all other bounds on each dimension remaining unchanged. The probabilities $P_f(F_1)$, $P_r^1(F_1)$ and $P_r^2(F_1)$ can all be expressed as a function with a single variable x. Given p_1 and p_2 in Figure 5, Figure 6 presents the probability functions of P_f, P_r^1 and P_r^2 on $x = F_1.l[k]$. Since p_2 is the dominator for p_1, the valid range of $x = F_1.l[1]$ is in the interval $[p_2^t[1], p_1^t[1]]$; otherwise there will be a violation of Lemma 4.2. $P_f(F_1)$ is a monotonically increasing function on x, since the increase of the lower bound will decrease the distance from p_1^t to the boundary. $P_r^1(F_1)$ is a monotonically decreasing function because a smaller lower bound allows the dominator to move away more easily. $P_r^2(F_1)$ is a step-wise constant function on different intervals, depending on the content of L. Since the update cost $C(F_1)$ is a weighted sum of the three different probabilities, it must be represented by a complicated function on x, where the optimal x is the global minimum. However, if we split the interval $[p_2^t[1], p_1^t[1]]$ into three subintervals, $[p_2^t[1], l_1[1]]$, $[l_1[1], l_3[1]]$ and $[l_3[1], p_1^t[1]]$, then on each sub-interval, the second-order derivative on the cost function is always positive. This implies that the local minimum in each interval can be found efficiently.

Algorithm 3 utilizes the above models for the construction of a robust filter set. The lower and upper bounds of each new filter are initialized to 0 and 1, respectively. Then, the server gradually shrinks the filters, following different methods for skyline and non-skyline records. Specifically, if r_i is in the skyline (Lines 4 to 10), two values V and Q ($V < Q$) are selected between r_i and every other record r_j on a chosen dimension k. V and Q are used to update the upper bound of F_i^t and lower bound of F_j^t respectively, to enforce Lemma 4.1. If r_i is not in the skyline (Lines 11 to 18), the server selects a dominator r_j for r_i and performs similar split operations on every dimension, enforcing Lemma 4.2. For simplicity, the pseudo-code does not distinguish whether r_j has a valid filter or not. In the former case, the algorithm uses $F_j^t[k]$ instead of $p_j^t[k]$ without affecting correctness. Note that filter construction is identical for both the initial and subsequent timestamps. The difference is that in the former case none of the records has a valid filter.

We clarify the selection of the splitting dimension k, and the values of V and Q in Lines 6 and 14 of Algorithm 3. Algorithm 4 iterates over every dimension, estimates the cost,

[3] C_i can be visualized as the maximum distance that p_i can move between two consecutive timestamps.

Algorithm 3 FilterConstruction (snapshot S_t)

1: Retrieve the recent probe history log L
2: **for** each record $r_i \in S_t$ without valid filter **do**
3:    Initialize F_i^t with $F_i^t.l[k]=0$ and $F_i^t.u[k]=1$ for all k
4: **for** each record $r_i \in Sky(S_t)$ without valid filter **do**
5:    **for** each record $r_j \in S_t$ $(j \neq i)$ **do**
6:       $(k, V, Q) =$ **SelectDimension**(r_i, r_j, L)
7:       **if** $F_i^t.u[k] > V$ **then**
8:          $F_i^t.u[k] = V$
9:       **if** r_j has no valid filter AND $F_j^t.l[k] < Q$ **then**
10:          $F_j^t.l[k] = Q$
11: **for** each record $r_i \notin Sky(S_t)$ without valid filter **do**
12:    Choose a dominator r_j for r_i
13:    **for** each dimension k **do**
14:       $(V, Q) =$ **SelectBoundary**(r_i, r_j, k, L)
15:       **if** $F_i^t.l[k] < V$ **then**
16:          $F_i^t.l[k] = V$
17:       **if** r_j has no valid filter AND $F_j^t.u[k] > Q$ **then**
18:          $F_j^t.u[k] = Q$
19: Return the filter set

Algorithm 4 SelectDimension (pair $\{r_i, r_j\}$, probe history log $L = \{l_1, l_2, \ldots, l_M\}$)

Set the current best cost C^* at infinity
Clear optimal dimension k^*, optimal upper bound V^* and optimal lower bound Q^*
for each dimension k **do**
   $(C, V, Q) =$**SelectBoundary**(r_i, r_j, k, L)
   if $C < C^*$ **then**
      Set $C^* = C$, $k^* = k$, $V^* = V$ and $Q^* = Q$
Return (k^*, V^*, Q^*)

and selects the one with the minimum cost. The details of the cost estimation and boundary selection on dimension k are summarized in Algorithm 5, which utilizes the observations of Figure 6. Given the set L of previous probe locations in $UDR(r_i)$, k is split into intervals based on the values of these probe sources on k. In each interval, the best boundary is computed using the monotonic derivative property. A greedy search strategy first decides the upper bound for F_i, while the new lower bound for F_j is selected later if r_j currently does not possess a valid filter F_j.

The filter set returned by Algorithm 3 is robust, but suboptimal. Next, we prove that optimal filter generation is intractable. Let r_0 be a skyline point, and $R = \{r_1, r_2, \ldots, r_n\}$ $(n > 1)$ be a set of records. For each $r_i \in R$ there should exist at least a dimension k $(1 \leq k \leq d)$ such that$(F_0.u[k] < r_i[k])$. This requirement is necessary for enforcing Lemma 4.1 independent of the filter construction algorithm[4]. Consider, for simplicity, that (i) R contains only non-skyline records, (ii) the co-ordinates of r_0 are 0 on each dimension, (iii) the value of each $r_i[k]$ is either 0 or 1 (since r_i is not in the skyline, at least one dimension should be 1), and (iv) if F_0 is bounded on a dimension k, then its extent on k is 0.5. $SF(r_0, R)$ denotes the problem of selecting the optimal F_0 in this setting. Figure 7 illustrates an example of $SF(r_0, \{r_1, r_2, r_3\})$, assuming d=3. The first filter is invalid because it covers p_1. Between the valid filters, the one in Figure 7(b) is better because it is unbounded on the third

[4]In Algorithm 3, this is implemented by Lines 2-10.

Algorithm 5 SelectBoundary (pair $\{r_i, r_j\}$, dimension k, probe history $L = \{l_1, l_2, \ldots, l_M\}$)

Construct $L_i = \{l_m[k] \mid l_m \in L \;\&\&\; l_m \in UDR(F_i)\}$
Split the interval $[p_i^t[k], F_j.l[k]]$ into at most $|L_i|+1$ sub-intervals, $\{I_1, \ldots, I_{|L_i|+1}\}$, with splits at each $l_m[k] \in L_i$
Set $C^* = C(F_i)$, $V^* = F_i.u[k]$
for each interval I_m **do**
   let V be the best dimension k value within I_m
   Construct F_i' by using V instead of $F_i.u[k]$
   if $C(F_i') < C^*$ **then**
      $C^* = C(F_i')$ and $V^* = V$
if F_j has no valid filter **then**
   Split the interval $[V^*, p_j^t[k]]$ into sub-intervals, $\{J_1, \ldots, J_{|L_i|+1}\}$ with splitting locations at $l_m[k] \in L_i$
   Set $C^* = C(F_j')$ and $Q^* = F_j.l[k]$
   for each interval J_m **do**
      Let Q be the best dimension k value in J_m
      Construct F_j' by using Q instead of $F_j.l[k]$
      if $C(F_j') < C^*$ **then**
         $C^* = C(F_j')$ and $Q^* = Q$
   Return $(C(F_j') + C(F_i'), V^*, Q^*)$
else
   Return $(C(F_j) + C(F_i'), V^*, F_j.l[k])$

dimension, and, therefore larger than that in Figure 7(c). In general, $SF(r_0, R)$ is equivalent to the problem of bounding F_0 on the minimum number of dimensions.

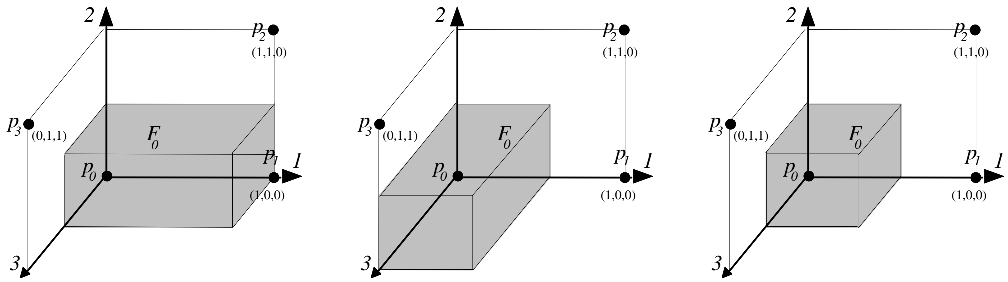
THEOREM 4.2. *$SF(r_0, R)$ is NP-hard.*

PROOF. We construct a polynomial reduction from the NP-Hard *hitting set* problem $HS(X, S)$, which is a variant of *set cover*. Given a set of items $X = \{x_1, x_2, \ldots, x_d\}$ and a collection $S = \{S_1, S_2, \ldots, S_n\}$ of subsets of X, a hitting set $H \subseteq X$ contains at least one item from each S_i. The goal of $HS(X, S)$ is to find the hitting set with the minimum cardinality. From an instance of $HS(X, S)$, we generate an instance of $SF(r_0, R)$ by converting each S_i to a record r_i, such that $r_i[k]$=1, if S_i contains item x_k; otherwise, $r_i[k]$=0. Given the optimal filter F_0, we obtain the minimal H by including only items that correspond to the bounded dimensions of the filter. $\square$

For example, if $X = \{x_1, x_2, x_3\}$, $S_1 = \{x_1\}$, $S_2 = \{x_1, x_2\}$, $S_3 = \{x_2, x_3\}$, then S_1, S_2, S_3 map to points p_1, p_2, p_3 in Figure 7. The optimal filter of Figure 7(b) is bounded on dimensions 1 and 2; therefore the corresponding minimal hitting set is $H = \{x_1, x_2\}$.

5. SAMPLING METHOD

By Theorem 3.1, any exact algorithm for FSQW must compute the skyline at each snapshot, potentially leading to high update cost. In this section, we introduce *Sampling*, which outputs approximate results of FSQW. In *Sampling*, at any time t, all clients collectively report their current status with some global probability R. In order to achieve this, the server initially sends the same message to each client. This message contains a random seed S and the sampling probability R. All clients run the same deterministic random number generator using S, and generate the same sequence $\{x_1, x_2, \ldots, x_t, \ldots\}$ $(0 \leq x_i \leq 1)$ with uniform distribution between 0 and 1. Each client will issue an update to the



(a) Invalid filter F_0 (b) Optimal filter F_0 (c) Sub-optimal filter F_0

Figure 7: Example of dimension selection during filter construction

Algorithm 6 Sampling(window size s, threshold θ)

1: Calculate the sampling probability R
2: Send probability R and random seed S to all clients
3: **for** each record r_i **do**
4: Clear the skyline counter $C_i = 0$
5: **for** each timestamp t **do**
6: **if** sampling snapshot S_t is scheduled **then**
7: Receive updates from all clients
8: Increase sample counter $T = T + 1$
9: $Sky(S_t)$ =**ComputeSkyline(**S_t**)**
10: **for** each $r_i \in Sky(S_t)$ **do**
11: Increase skyline counter $C_i = C_i + 1$
12: **if** snapshot S_{t-s} was previously sampled **then**
13: Decrease sample counter $T = T - 1$
14: **for** each $r_i \in Sky(S_{t-s})$ **do**
15: Decrease skyline counter $C_i = C_i - 1$
16: Output all records with $C_i \geq \theta \cdot T$

server at timestamp t, if $x_t \leq R$. Clients entering the system at later times can synchronize[5] with the rest by obtaining (from the server) the number of timestamps that have elapsed since initialization.

Algorithm 6 summarizes $FSQW$ processing at the server. The number of sampled snapshots in the current window is stored in the sample counter T. Each record r_i in the system has a skyline counter C_i that keeps the skyline frequency of r_i on the sampled snapshots. The server also maintains buffers with the skyline set on each sampled snapshot. At time t, if the current snapshot S_t is scheduled to be sampled, the algorithm computes the skyline and increments the counter C_i of each skyline record r_i, as well as T. Lines 12-15 expunge the expiring timestamp S_{t-s} (recall that the current window W_t^s contains timestamps $t - s + 1$ to t). Specifically, if S_{t-s} was previously sampled in the system, the server decrements T and the counters for skyline records in $Sky(S_{t-s})$. Finally, records with skyline counter exceeding $\theta \cdot T$ are reported as results of $FSQW(\theta, W_t^s)$.

The sampling probability R determines the trade-off between accuracy and communication overhead. Given predefined values for the error tolerance ϵ and confidence δ, we provide guidelines for the choice of R values.

[5]In general, *Sampling* requires a synchronous communication protocol, whereas in *Filter* clients can issue asynchronous updates (i.e., whenever there are filter violations).

LEMMA 5.1. *If we have* $\frac{2\ln(1/\delta)}{\epsilon^2\theta}$ *sampled snapshots in the current window* W_t^s*, any point reported by Algorithm 6 at timestamp* t *has skyline frequency larger than* $(\theta-\epsilon)s$ *with probability* $1-\delta$.

This lemma is an easy extension of the Chernoff bound [11] and implies that the $FSQW$ result is robust, if we can guarantee that the number of sampled snapshots in every sliding window exceeds $\frac{2\ln(1/\delta)}{\epsilon^2\theta}$.

THEOREM 5.1. *When* $R \geq \sqrt{\frac{ln(1/\delta)}{2s}} + \frac{2\ln(1/\delta)}{\epsilon^2\theta s}$*, any point reported by Algorithm 6 at any time has skyline frequency larger than* $(\theta-\epsilon)s$ *with probability* $1-2\delta$

PROOF. In the first part of the proof, we want to show that the probability of having more than $\frac{2\ln(1/\delta)}{\epsilon^2\theta}$ sampled snapshots is larger than $1-\delta$ in any sliding window. This is proven by using Chebychev's inequality of binomial distribution. If X is the number of sampled snapshots, we have $\Pr(X \leq x) \leq \exp\left(-\frac{2(sR-x)^2}{s}\right)$

By replacing x with $\frac{2\ln(1/\delta)}{\epsilon^2\theta}$ and R with $\sqrt{\frac{ln(1/\delta)}{2s}} + \frac{2\ln(1/\delta)}{\epsilon^2\theta s}$, the probability is less than δ. Thus, there is $1-\delta$ probability to get enough samples.

The second part of the proof is based on the correctness of the frequent skyline points. By applying Lemma 5.1, the probability of outputting true frequent skyline point is at least $1-\delta$. Therefore, the probability of both above events happening at the same time is $(1-\delta)*(1-\delta) \geq 1-2\delta$. This completes the proof of the theorem. $\square$

Given the input requirements on the error rate ϵ and the confidence δ, the minimum acceptable sampling rate can be calculated using Theorem 5.1. The sampling algorithm thus employs this sampling rate to guarantee the accuracy of the continuous skyline results.

6. HYBRID METHOD

Filter exploits the fact that often updates are gradual and infrequent. Subsequently, it achieves significant savings for records that exhibit these properties. However, highly dynamic objects, involving abrupt and/or very frequent updates, incur a large number of filter failures, which may cancel its advantage. *Sampling*, on the other hand, is oblivious to the update characteristics, implying that it may incur unnecessary uplink messages for records that are rather stable.

Motivated by these observations, we propose *Hybrid*, an approximate algorithm integrating filtering and sampling in a common framework.

In *Hybrid*, each record is in one of the following modes: *filter mode* (FM), *sampling mode* (SM) or *mixed mode* (MM). For records in FM or MM, filters are constructed and maintained according to *Filter*, i.e., at each timestamp the server receives failure messages and transmits probe requests. For records in SM or MM, snapshots are sampled according to *Sampling*. The motivation behind MM is that skyline records are important for the accuracy of FSQW results. Thus, MM ensures that they are regularly sampled, even if they are in FM mode[6]. Non-skyline records are either in FM or SM, depending on their estimated update cost to be discussed shortly.

Hybrid switches frequently updated non-skyline records to SM so that they issue updates only at the sampled timestamps (instead of every filter violation). On the other hand, relatively stable points remain in FM, so that they are not regularly sampled before they can influence the skyline. Next, we discuss the transition from FM to SM. The server keeps the number of messages related to each record r_i in FM. When this number exceeds the sampling rate R by a predefined factor (in our implementation we use $2R$), r_i is switched to SM, in order to reduce the transmissions. The corresponding client will receive the random seed sent by the server to start the synchronized sampling process.

If r_i is in SM, the server needs to estimate the transmission cost if r_i were in FM. This is accomplished by simulating a virtual filter over r_i, which is updated at every sampling timestamp according to the current locations of the objects in FM. If the filter does not need any update after a number of consecutive sampling timestamps (in our implementation we use 2), the server switches r_i to FM.

Algorithm 7 presents the general framework of *Hybrid* for processing $FSQW(\theta, W_t^s)$. *Hybrid* outputs approximate results based on the sampled snapshots, following the randomized strategy of *Sampling*. However, the filters for the set P of records in FM and MM must be updated at every timestamp, in order to set the appropriate mode for each record. This update follows Algorithm 1, which requires the computation of the partial skyline on P. Figure 8 shows an example where r_6 is in FM, r_2, r_5 and r_7 are in SM, while r_1, r_3 and r_4 are in MM. Given the current setting of the modes, the filters are constructed based only on r_1, r_3, r_4 and r_6. Compared to Figure 1, the filters are enlarged. On the other hand, since r_2 is in sampling mode, there is no filter update, even if r_2 moves into filters F_1 or F_3.

LEMMA 6.1. *Algorithm 7 outputs the correct skyline set at each sampling snapshot.*

PROOF. Any record in FM cannot be in the skyline because there must be some skyline point in MM dominating it and bounding its filter. On the other hand, all tuples not in FM must issue updates at each sampling timestamp. Thus, the skyline of records in MM and SM is the correct skyline for the entire snapshot. □

Since the records in MM and SM are regularly sampled, the above theorem implies that the results of FSQW reported by *Hybrid* must be the same as that of *Sampling*, leading to the following corollary of Theorem 5.1.

[6]Note that a skyline record in SM does not have to be in FM.

Algorithm 7 Hybrid (window size s, threshold θ)

Calculate the sampling probability R
Send probability R and random seed S to all clients
for each record r_i **do**
  Clear skyline counter $C_i = 0$
  Set r_i in FM
for each timestamp t **do**
  Receive updates from clients in FM or MM due to filter violations
  if sampling snapshot S_t is scheduled **then**
    Receive updates from clients in SM
    Increase sample counter $T = T + 1$
    $Sky(S_t)$ =**ComputeSkyline**(S_t)
    for each $p_i \in Sky(S_t)$ **do**
      Increase skyline counter $C_i = C_i + 1$
  Construct a set P with all records in FM or MM
  $Sky(P)$ =**ComputeSkyline**(P)
  FilterConstruction($Sky(P)$)
  if snapshot S_{t-s} was previously sampled **then**
    Decrease the sample counter $T = T - 1$
    for each $r_i \in Sky(S_{t-s})$ **do**
      Decrease the skyline counter $C_i = C_i - 1$
  Switch the modes of the clients if necessary
  Output all records with $C_i \geq \theta T$

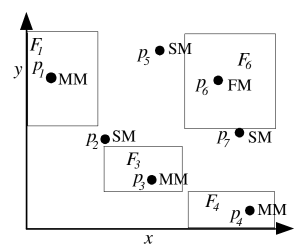


Figure 8: Example of *Hybrid*

COROLLARY 1. *If the sampling rate* $R \geq \sqrt{\frac{ln(1/\delta)}{2s} + \frac{2\ln(1/\delta)}{\epsilon^2 \theta s}}$, *any frequent skyline point reported by Hybrid is in the skyline for at least* $(\theta - \epsilon)s$ *snapshots with confidence* $1 - 2\delta$.

7. EXPERIMENTS

This section evaluates experimentally the proposed methods and compares them against *Naive*, a baseline algorithm that transmits all updates to the server without incurring downlink messages. We only include FSQW queries because, as discussed in Section 3, snapshot skylines constitute a special case of FSQW where s and θ equal 1. All experiments are executed on a PIII 1.8GHz CPU, with 1GB main memory. The programs are compiled by GCC 3.4.3 in Linux. Section 7.1 compares the algorithms on synthetic data, and Section 7.2 on real data sets.

7.1 Synthetic Data

The synthetic data sets are created using the standard generator [3] with three common distributions: independent(I), correlated(C) and anti-correlated(A). Every attribute on each dimension is a real number between 0 and 1. We introduce updates on the generated data set by applying an uncertainty parameter u. Specifically, an object is allowed to move within the space defined by a rectangle with edge

Parameter	Range
Dimensionality	2,3,**4**,5,6
Distribution	C,**I**,A
Data Size (K)	2.5,**5**,10,20
Threshold θ	0.6,0.7,**0.8**,0.9
Error rate ϵ	0.1,**0.2**,0.3,0.4
Confidence δ	0.05,**0.1**,0.15,0.2
Uncertainty u	0.01,0.02,0.04,**0.08**,0.16

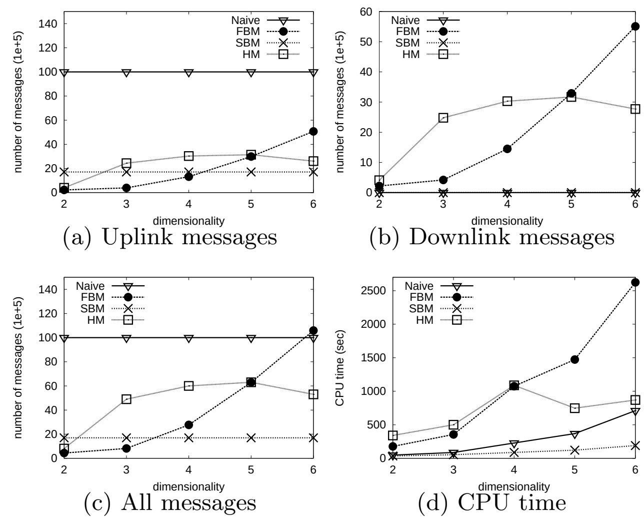
Table 1: Experimental parameters

extent $2u$ and center at the original position. At each timestamp t, the locations of the objects follow uniform distribution in their corresponding rectangles. Table 1 illustrates the range and default values (in bold) of the parameters involved in our experiments. In each experiment, we vary a single parameter, while setting the rest to their default values. Recall that ϵ, δ are used to tune the sampling rate, and are applicable only to *Sampling* and *Hybrid*. For all experiments, we set the window size of FSQW queries to 1000 (i.e., a query returns the frequent skyline points in the last 1000 timestamps).

We evaluate the performance of the algorithms on six measures. The first three refer to the number of uplink, downlink, and total messages. Assuming that the uplink (c_u) and downlink (c_d) costs are equal, the total number of messages reflects the overall transmission cost. The fourth measure is the CPU time. The above measures assess the efficiency of the algorithms. The last two measures, recall and precision, assess the quality of the query results, averaged over all timestamps. Specifically, if A_t is the result of algorithm A and FS_t is the correct result at time t, recall is defined as the ratio $|A_t \cap FS_t|/|FS_t|$ and precision as $|A_t \cap FS_t|/|A_t|$. Precision and recall are only measured for *Sampling* and *Hybrid* because *Filter* produces the exact results.

Figure 9 presents the efficiency measures as a function of dimensionality for the independent distribution. We use the acronym FBM for *Filter*, SBM for *Sampling* and HM for *Hybrid*. In 2D space, FBM outperforms the other methods in terms of transmission cost due to its advantage on the number of uplink messages. However, its overhead increases dramatically with the dimensionality, and when d reaches 6, it is outperformed even by *Naive*. SBM has the best overall behavior (in terms of both communication overhead and CPU cost) since it is independent of the dimensionality. HM requires fewer messages than SBM in the 2D dataset, but in general its performance lies between that of SBM and FBM. Although *Naive* does not incur any downlink messages, it is the worst method for the communication overhead in all but one settings.

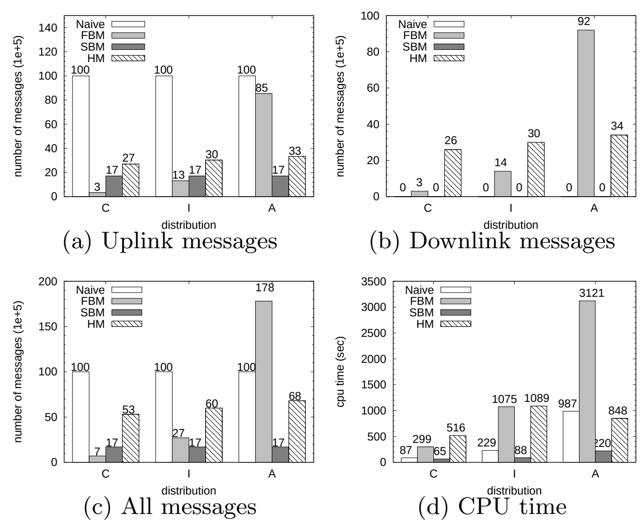
Figure 10 tests the algorithms on different types of distributions. FBM has a clear advantage when the records are positively correlated and the skyline cardinality is small. However, if the distribution is anti-correlated, most of the object are in skyline. Consequently, the filter updates are so frequent that (almost) each record incurs one uplink message due to filter failure and one downlink message for the new filter per timestamp. The update cost of HM is stable even on anti-correlated distributions, in which case most of the objects are in sampling mode, instead of filter or mixed mode. SBM has again good overall performance in terms of both network and CPU overhead.



(a) Uplink messages (b) Downlink messages

(c) All messages (d) CPU time

Figure 9: Efficiency vs. dimensionality (ind.)



(a) Uplink messages (b) Downlink messages

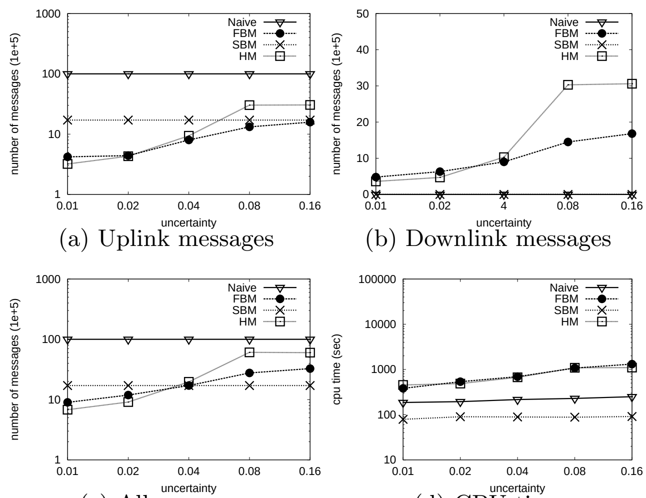
(c) All messages (d) CPU time

Figure 10: Efficiency vs. distribution (4D)

In Figure 11, we vary the uncertainty parameter u on the synthetic data sets. For low values of u, objects move within a small range of the underlying space. FBM and HM take advantage of the locality property, since the records are usually bounded by a stable filter. The update cost of HM is sometimes lower than that of FBM when the uncertainty is below 0.2, as it switches the more uncertain objects to sampling mode. However, the CPU costs of FBM and HM are still much worse than that of SBM.

Figure 12 evaluates the efficiency measures as a function of the number of records in the system. Our methods scale better than *Naive*, whose transmission overhead is linear to the data cardinality. The CPU cost has quadratic complexity because of the dominance checks (in all algorithms) and the filter computations (in FM, HM).

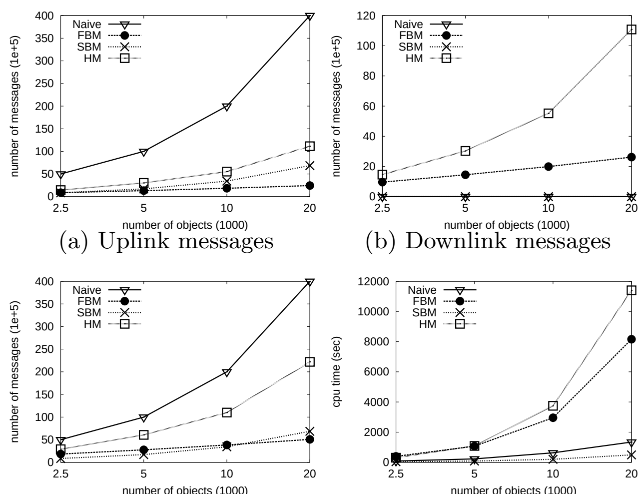
Next, we focus on the recall and precision of SBM and HM. By Theorem 5.1 and Corollary 1, the sampling rate of SBM and HM is decided by the error rate ϵ and the con-



(a) Uplink messages (b) Downlink messages

(c) All messages (d) CPU time

Figure 11: Efficiency vs. uncertainty (ind. 4D)



(a) Uplink messages (b) Downlink messages

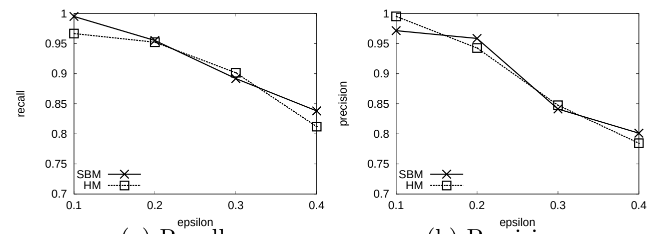
(c) All messages (d) CPU time

Figure 12: Efficiency vs. data cardinality (ind. 4D)

fidence δ. Furthermore, the desired frequency θ of points in the FSQW result is related to the sampling rate. Thus, we test the impact of these parameters on the recall and precision. Due to the random nature of the output, we perform each experiment three times and report the average measurements.

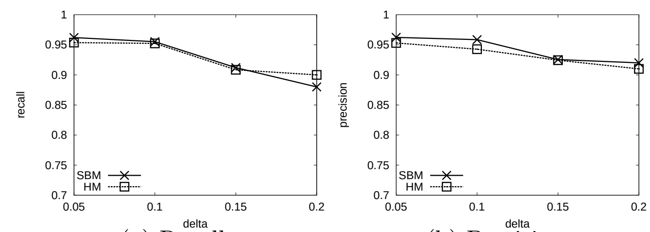
Figure 13 shows the effect of the error rate ϵ on the quality measures. When ϵ is up to 0.2 (the default value), the recall and precision of both methods exceed 0.9. Even when ϵ reaches 0.4, the quality of the result is still acceptable with recall and precision above 0.8. As shown in Figure 14, the impact of δ on the result quality is not as large as that of ϵ. SBM and HM have similar behavior in all cases because they apply the same values of ϵ and δ.

Figure 15 evaluates the effect of the query threshold θ, which bounds the frequency of the points to be included in FSQW. When $\theta = 0.7$, the quality of the query output is much worse than other values because there are many



(a) Recall (b) Precision

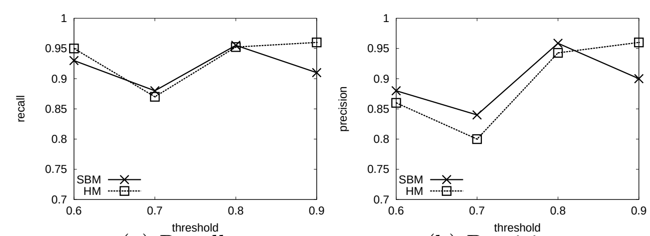
Figure 13: Quality vs. error rate (ind. 4D)



(a) Recall (b) Precision

Figure 14: Quality vs. confidence (ind. 4D)

objects in the system with skyline frequency around 0.7 over the sliding windows. The sampling rate fails to distinguish the actual results leading to larger error.



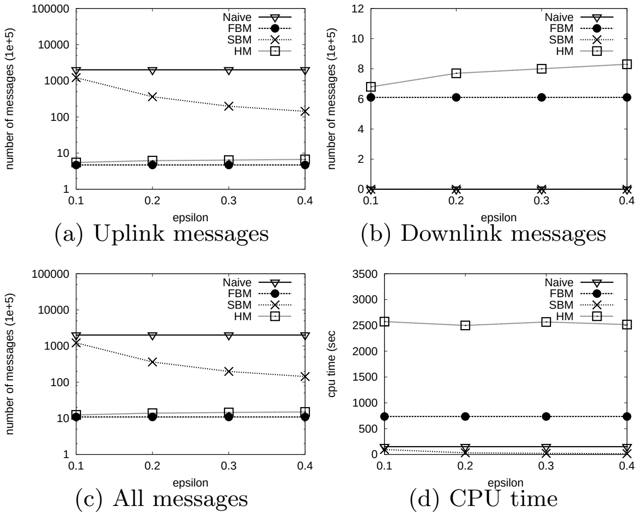
(a) Recall (b) Precision

Figure 15: Quality vs. threshold (ind. 4D)

7.2 Real Data

The real data were collected by Lawrence Berkeley Laboratory, and contain a 30-day trace of the TCP connections between their local network and the internet[7]. The remote IP addresses in all connections are divided into groups, according to the first 24 bits of their IPs. For example, "172.18.179.20" and "172.18.179.38" are in the same group, while "172.18.180.22" is not. The connections are classified into four categories based on their protocol type: NNTP, TCP-DATA, SMTP and OTHERS. By taking the snapshot every 100 seconds, an address group G_i dominates another group G_j at time t, if G_i has no fewer connections than G_j on all four types in the last 1000 seconds and more on at least one category. The skyline contains the non-dominated groups. Given the original data set with 782281 connections recorded in 2591987 seconds, we transform it into a new 4-dimensional data set with 25920 snapshots and 7776 address groups. Since the data characteristics are fixed, we

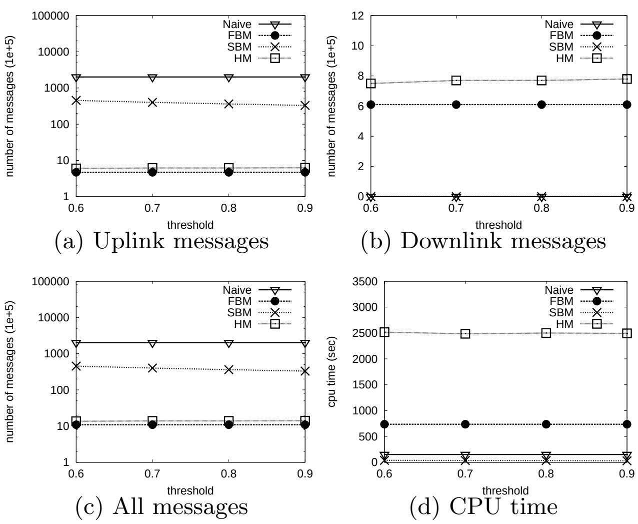
[7]http://ita.ee.lbl.gov/html/contrib/LBL-CONN-7.html



(a) Uplink messages (b) Downlink messages

(c) All messages (d) CPU time

Figure 16: Efficiency vs. error rate (TCP)



(a) Uplink messages (b) Downlink messages

(c) All messages (d) CPU time

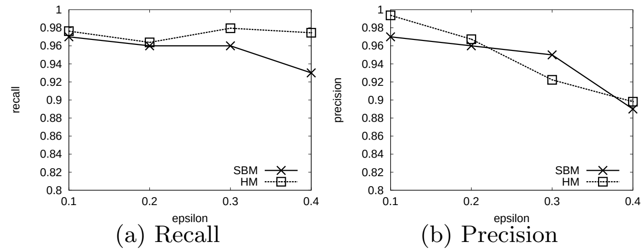
Figure 17: Efficiency vs. threshold (TCP)

cannot evaluate factors such as distribution, dimensionality and cardinality.

Figure 16 measures efficiency as a function of the error rate. FBM and HM incur less communication cost than *Naive* and SBM. When $\epsilon = 0.1$, the overhead of SBM is almost equal to that of *Naive*, suggesting that it samples each record almost on a per-timestamp basis. FBM and HM are better by about two orders of magnitude. However, they incur significantly higher CPU cost due to the heavy computation on filter updates.

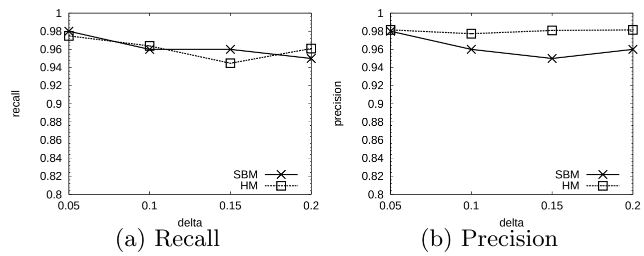
Figure 17 summarizes the efficiency results for varying the frequency threshold θ over TCP. The impact of θ on the performance of the algorithms is negligible. This phenomenon stems from the properties of the TCP data. Specifically, a selective set of IP address groups have very high skyline frequencies, while the rest rarely appear in the skyline. Thus, the change on the threshold hardly affects the query result of FSQW and the performance of all methods.

Figures 18 and 19 display the recall/precision as functions of the error rate ϵ and the confidence δ. As in the synthetic data sets, the quality of SBM and HM is high and both algorithms achieve values above 0.9.



(a) Recall (b) Precision

Figure 18: Quality vs. error rate



(a) Recall (b) Precision

Figure 19: Quality vs. confidence

Summarizing the experimental evaluation, FBM is the best method for lower dimensionality and correlated data sets. On the other hand, SBM is usually the method of choice for high dimensions and anti-correlated data. HM has a balanced behavior in between FBM and SBM. All algorithms outperform *Naive*, usually by large margins. Finally, SBM and HM provide high-quality results and are rather insensitive to the choices of ϵ and δ. IN addition, the sampling rate provides a mechanism for tuning the trade-off between accuracy and overhead; e.g, a low sampling rate can be used for applications that need to minimize the cost at the expense of result quality.

8. CONCLUSION

Snapshot skylines change too fast to be meaningful in streaming environments. Instead, it is more interesting to identify the records that consistently appear in the skyline over several timestamps. Motivated by this observation, we introduce the concept of *frequent skyline query over a sliding window*. The output of FSQW is the set of records that appear in the skylines of at least θ of the s most recent timestamps. We propose three algorithms for minimizing the communication overhead of FSQW processing: *Filter*, *Sampling* and *Hybrid*.

Filter avoids transmission of updates from objects that cannot influence the skyline. Specifically, the server computes, for each record, a hyper-rectangle that bounds the value of each attribute. The corresponding client needs to issue an update only if the record violates its filter, or if the server explicitly asks for the current attribute values. In *Sampling* the clients transmit updates at a rate that depends on the desired trade-off between accuracy and overhead (but is independent of the dataset). *Hybrid* integrates *Filter* and *Sampling*, allowing records to switch among three different modes depending on their properties.

9. ACKNOWLEDGEMENT

Zhenjie Zhang and Dimitris Papadias were supported by grant 6184/06 from Hong Kong RGC. Reynold Cheng was supported by grants 5135/08 and 5133/07 by Hong Kong RGC, the Germany/HK Joint Research Scheme (Project G_HK013/06), and the University of Hong Kong (Project 200808159002). Zhenjie Zhang and Anthony K.H. Tung were supported by Singapore ARF grant R-252-000-268-112. We thank the reviewers for their insightful comments.

10. REFERENCES

[1] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD Conference*, pages 28–39, 2003.

[2] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004.

[3] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[4] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, pages 203–214, 2005.

[5] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD Conference*, pages 503–514, 2006.

[6] R. Cheng, B. Kao, S. Prabhakar, A. Kwan, and Y.-C. Tu. Adaptive stream filters for entity-based queries with non-value tolerance. In *VLDB*, pages 37–48, 2005.

[7] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[8] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast protocols to support efficient retrieval from databases by mobile users. *ACM Trans. Database Syst.*, 24(1):1–79, 1999.

[9] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.

[10] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, pages 229–240, 2005.

[11] T. Hagerup and C. Rüb. A guided tour of chernoff bounds. *Inf. Process. Lett.*, 33(6):305–308, 1990.

[12] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD Conference*, pages 479–490, 2005.

[13] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline queries against mobile lightweight devices in manets. In *ICDE*, page 66, 2006.

[14] A. Jain, E. Y. Chang, and Y.-F. Wang. Adaptive stream resource management using kalman filters. In *SIGMOD Conference*, pages 11–22, 2004.

[15] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[16] K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in z order. In *VLDB*, pages 279–290, 2007.

[17] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005.

[18] M. D. Morse, J. M. Patel, and W. I. Grosky. Efficient continuous skyline computation. In *ICDE*, page 108, 2006.

[19] M. D. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278, 2007.

[20] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao. A threshold-based algorithm for continuous monitoring of k nearest neighbors. *IEEE Trans. Knowl. Data Eng.*, 17(11):1451–1464, 2005.

[21] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD Conference*, pages 563–574, 2003.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[23] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[24] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. Computers*, 51(10):1124–1140, 2002.

[25] M. Sharifzadeh and C. Shahabi. The spatial skyline queries. In *VLDB*, pages 751–762, 2006.

[26] K. L. Tan, P. K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.

[27] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *TKDE*, 18(3):377–391, 2006.

[28] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu. Efficient skyline query processing on peer-to-peer networks. In *ICDE*, pages 1126–1135, 2007.

[29] P. Wu, D. Agrawal, Ö. Egecioglu, and A. E. Abbadi. Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In *ICDE*, pages 486–495, 2007.

[30] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi. Parallelizing skyline queries for scalable distribution. In *EDBT*, pages 112–130, 2006.

[31] T. Xia and D. Zhang. Refreshing the sky: the compressed skycube with efficient support for frequent updates. In *SIGMOD*, pages 491–502, 2006.

[32] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB*, pages 241–252, 2005.