다중 연속 스카이라인 질의의 효율적인 처리 기법

Multiple Continuous Skyline Query Processing Over Data Streams

이유원(Yu Won Lee)*, 이기용(Ki Yong Lee)**, 김명호(Myoung Ho Kim)***

초 록

최근 들어 e-비즈니스 환경에서도 증권 거래, 시세, 주문 및 과금 데이터와 같이 지속적으로 유입되는 데이터 스트림에 대한 처리가 중요해지고 있다. 이 중에서도 데이터 스트림에 대한 다기준 의사 결정에 사용되는 스카이라인(skyline) 질의의 사용이 증가하고 있다. 다차원 튜플의 집합이 주어졌을 때, 스카이라인 집합은 다른 튜플에 의해 지배(dominate)되지 않는 튜플들의 집합을 반환한다. 고정된 데이터에 대한 단일 스카이라인 질의 처리에 대해서는 최근까지 많은 연구가 이루어져 왔으나, 데이터 스트림 환경에서 다중 연속 스카이라인 질의 처리에 대해서는 아직까지 많은 연구가 수행되지 않았다. 본 논문에서는 데이터 스트림 환경에서 하나 이상의 연속 스카이라인 질의들이 주어졌을 때, 이들을 효율적으로 처리할 수 있는 방법을 제안한다. 제안하는 방법은 각 튜플이 어떤 질의의 결과에 포함될지를 효율적으로 파악함으로써, 여러 개의 연속 스카이라인 질의들도 적은 비용으로 동시에 처리할 수 있다. 다양한 실험을 통해 제안하는 방법의 우수성을 보인다.

ABSTRACT

Recently, the processing of data streams such as stock quotes, buy-sell orders, and billing records becomes more important in e-Business environments. Especially, the use of skyline queries over data streams is rapidly increasing to support multiple criteria decision making. Given a set of multi-dimensional tuples, a skyline query retrieves a set of tuples which are not dominated by other tuples. Although there has been much work on processing skyline queries over static datasets, there has been relatively less work on processing multiple skyline queries over data streams. In this paper, we propose an efficient method for processing multiple continuous skyline queries over data streams. The proposed method efficiently identifies which tuple is a skyline tuple of which query, resulting in a lower cost of processing multiple skyline queries. Through performance evaluation, we show the performance advantage of the proposed method.

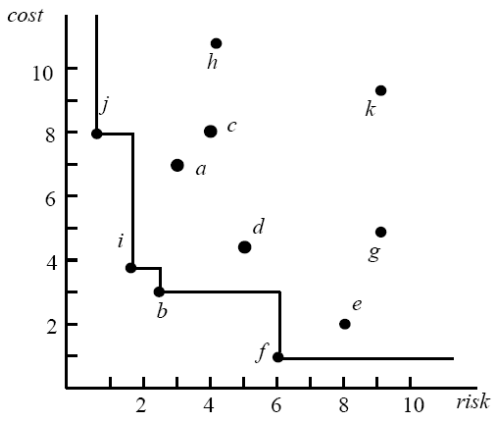
키워드 : 다중 스카이라인 질의, 데이터 스트림, 연속 질의
Multiple Skyline Queries, Data Streams, Continuous Queries

* 주저자, KAIST 전산학과 박사과정
** 교신저자, 숙명여자대학교 컴퓨터과학과 조교수
*** 공저자, KAIST 전산학과 교수(Professor, Department of Computer Science, KAIST)
2010년 09월 27일 접수, 2010년 10월 17일 심사완료 후 2010년 11월 05일 게재확정.

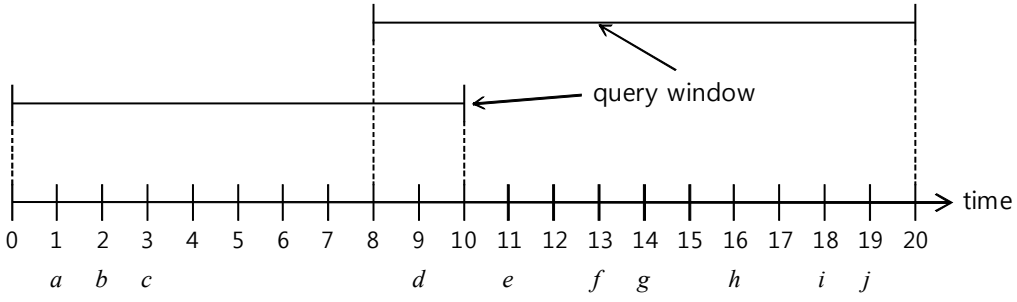
1. 서 론

최근 들어 e-비즈니스 환경에서도 증권 거래 시스템에서의 시세, 주문 및 과금 데이터와 같이 지속적으로 끊임없이 유입되는 데이터 스트림에 대한 처리가 매우 중요한 연구 주제가 되고 있다. 이 중에서도 특히 다차원 데이터가 유입되는 데이터 스트림에 대해, 다기준(multiple criteria) 의사결정에 사용되는 스카이라인(skyline) 질의의 처리가 날로 중요해지고 있다. d-차원 튜플의 집합이 주어졌을 때, 스카이라인 질의는 다른 튜플에 의해 지배(dominate)되지 않는 튜플들의 집합을 반환한다. 어떤 튜플 t가 다른 튜플 t'보다 최소한 하나 이상의 차원에 대해서는 더 우월하고, 다른 모든 차원들에 대해서도 적어도 열등하지 않을 때, t는 t'를 지배한다고 한다. 본 논문에서는 일반성을 잃지 않으면서, 어떤 튜플 t가 다른 튜플 t'보다 어떤 차원에서의 값이 작을 때, t가 t'보다 해당 차원에서 더 우월하다 라고 말한다. <그림 1>은 2차원 튜플의 집합에 대한 스카이라인의 예를 보여준다. <그림 1>에서 x축은 주식의 위험도를, y축은 가격을 나타낸다. <그림 1>에서 튜플 j, i, b, f는 다른 튜플에 의해 지배되지 않기 때문에 스카이라인에 속한다. 이들은 다른 주식에 비해 위험도가 낮거나 혹은 가격이 낮은 주식을 뜻한다. 한편, j, i, b, f는 서로 지배하는 관계에 있지 않다.

고정된 튜플의 집합에 대한 스카이라인 질의 처리에 대해서는 지금까지 많은 연구가 이루어져 왔다[8, 2, 3, 4, 5, 9, 13, 7, 11]. 반면에 데이터 스트림 환경에서는 튜플의 집합이 고정되어 있지 않고, 튜플들이 지속적으로 유입된다. 데이터 스트림에 대해 지속적으로 결과를 반환하는 질의를 연속 질의라고 한다. 데이터 스트림 환경에서는 데이터 객체들이 무한히 유입될 수 있으므로, 데이터 스트림에 대한 질의에는 일반적으로 슬라이딩 윈도우(sliding window)가 정의된다. 슬라이딩 윈도우는 윈도우의 길이를 나타내는 인자 RANGE와 윈도우의 이동 간격의 크기를 나타내는 인자 SLIDE로 정의된다[1]. <그림 2>는 슬라이딩 윈도우가 RANGE = 12와 SILDE = 10인 스카이라인 질의의 처리 예를 보여준다. 본 논문에서 시간 단위는 초(second)라고 가정한다. <그림 2>은 튜플 a, b, c, d, …, j의 도착시간이 각각 1, 2, 3, 9, …, 19임을 나타낸다. <그림 2>에서 현재 시간이 10일 때, 해당 스카이라인 질의는 시간 구간[-2, 10] = [0, 10] 내에 도착한 튜플, 즉 {a, b, c, d, e}에 대한 스카이라인을 출력한다. 10초 후에 해당 질의는 시간 구간[8, 20] 내에 도착한 튜플, 즉 {d, e, f, g, h, i, j}에 대한 스카이라인을 출력한다. 여기서 스카이라인을 출력해야 하는 각각의 시간 구



〈그림 1〉 스카이라인의 예



〈그림 2〉 연속 스카이라인 질의 처리의 예

간을 질의 윈도우(query window)라고 한다. 데이터 스트림 환경에서의 연속 스카이라인 질의를 처리하는데 있어, 고정된 데이터를 가정하고 있는 기존의 스카이라인 질의 처리 기법은 적용이 어렵거나 매우 비효율적이다. 최근에 [14]는 데이터 스트림 환경에서 단일 연속 스카이라인 질의에 대한 처리 기법을 제안했다. 하지만 해당 방법은 단일 질의에 대한 것이며, 슬라이딩 윈도우가 SLIDE = 1인 경우만을 고려하고 있다. 본 논문에서는 데이터 스트림 환경에서 여러 개의 연속 스카이라인 질의들이 주어졌을 때, 이들을 모두 효율적으로 처리할 수 있는 방법을 제안한다. 본 논문에서 제안하는 방법에서는 각 스카이라인 질의가 서로 다른 RANGE와 SLIDE를 가질 수 있다. 제안하는 방법은 새로운 튜플이 도착할 때마다, 해당 튜플이 어떤 질의의 결과에 포함될지에 대한 정보를 효율적으로 파악한다. 제안하는 방법은 이러한 정보를 이용하여 각 스카이라인 질의의 결과를 매우 적은 비용으로 얻어낼 수 있다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구를 소개한다. 제 3장에서는 본 논문에서 다루는 문제를 기술하고, 데이터 스트림 환경에서 다중 연속 스카이라인 질의를 처리하는 단순한 방법을 소개한다. 제 4장에서는 본 논문에서 제안하는 다중 연속 스카이라인 질의 처리 기법을 기술한다. 제 5장에서는 제안 방법에 대한 실험 성능 평가 결과를 제시하고, 마지막으로 제 6장에서는 본 논문의 결론을 맺는다.

2. 관련 연구

스카이라인 질의 처리에 관해서는 지금까지 많은 연구가 있어 왔다. 스카이라인의 계산은 [8]에서 처음 제안되었으며, [2]에서는 다차원 데이터의 수가 n이고, 각 차원의 값 분포가 독립일 때 $O(n)$의 시간 복잡도로 스카이라인을 구할 수 있는 방법을 제안하였다.

[3]은 데이터베이스 환경에서의 스카이라인 질의를 정의하고 그의 SQL 문법을 제안하였다. 그와 함께 [3]은 스카이라인을 계산하는 두 가지 알고리즘, 즉 BNL(block-nested-loop) 방법과 DC(divide-and-conquer) 방법을 제안하였다. [4]는 사전 정렬 기법을 이용하여 BNL 방법을 개선한 SFS(sort-filter

-skyline) 방법을 제안하였으며, [5]는 SFS 방법을 더욱 개선하였다. [13]은 전체 데이터 중 일부만 보고도, 부가적인 정보를 사용하여 스카이라인 결과를 점진적으로 출력하는 방법을 제안하였으며, [7]은 최근접 이웃 탐색(nearest neighbor search) 기법에 기반한 점진적인 스카이라인 계산 방법을 제안하였다. [11]에서는 데이터들이 R-tree로 색인되었을 경우, 해당 R-tree를 분기한정법(branch-and-bound)으로 탐색하여 스카이라인을 구하는 알고리즘(BBS)을 제안하였다.

하지만 지금까지 소개한 스카이라인 계산 기법들은 모두 고정된 데이터에 대한 것이며, 계산 전에 모든 데이터가 주어져 있다고 가정한다. 이에 반해 데이터 스트림 환경에서는 데이터가 미리 주어져 있지 않으며 지속적으로 유입된다. 이러한 환경에서는 매번 튜플이 도착할 때마다 스카이라인을 새로 계산하는 것은 현실적으로 불가능하며, 따라서 고정된 데이터를 가정하는 기존의 방법들은 매우 비효율적이거나 적용이 불가능하다.

[14]와 [10]은 데이터 스트림 환경에서의 스카이라인 질의 처리를 연구하였다. [14]는 튜플이 새로 도착할 때마다 기존 스카이라인 질의의 결과를 점진적으로 갱신한다. 이를 위해 해당 방법은 메모리에 두 집합 DB_{sky}와 DB_{rest}를 유지한다. DB_{sky}는 현재의 스카이라인 질의 결과를 담고 있으며, DB_{rest}는 현재는 스카이라인 질의 결과에 포함되지 않지만 추후에 포함될 가능성이 있는 튜플들을 담고 있다. 해당 방법은 새로운 튜플이 도착하거나 기존의 튜플이 만료되어 윈도우에서 제거될 때마다 DB_{sky}와 DB_{rest}를 점진적으로 갱신한다. 하지만 이 방법은 단일 연속 스카이라인 질의의 처리를 위한 것이며, SLIDE = 1인 슬라이딩 윈도우만을 고려하고 있다. [10]은 가장 최근의 $n(\forall n \leq N)$ 튜플들에 대한 스카이라인을 계산하는 n-of-N 스카이라인 질의에 대한 처리 기법을 제안하였다. 해당 방법은 인코딩 기법과 탐색 공간 축소(pruning) 기법을 사용하여 메모리 사용량을 줄인다. 하지만 이 방법은 모든 스카이라인 질의에 대해 동일한 N이 미리 주어져 있어야 한다는 제약이 있다. [12]는 분산 데이터 스트림 환경에서 스카이라인 계산을 효율적으로 계산하는 방법을 연구했다. 이 방법은 데이터 소스들 간의 통신비용을 줄이기 위해 최소의 정보만 전송한다. 본 논문에서 제안하는 방법은 서버에서 스카이라인 질의를 처리할 때 계산 비용을 줄이는 데 초점을 두고 있기 때문에 [12]의 연구와는 다르다. [6]은 데이터스트림 환경에서 부분차원 스카이라인 질의를 처리하는 방법에 대해 연구했다. 본 연구는 부분차원이 아닌 모든 차원에 대한 스카이라인 질의 처리를 다루므로 [6]과는 다르다.

3. 예비 지식

본 장에서는 먼저 본 논문에서 다루는 문제를 기술하고, 다중 연속 스카이라인 질의를 처리하는 가장 단순한 방법 두 가지를 소개한다.

3.1 문제 기술

데이터 스트림을 S라 하자. d-차원 공간을 R^d라 했을 때, d-차원 튜플들이 지속적으로

유입되는 데이터 스트림 S는 다음과 같이 표현된다.

$S = \{<t, arrival\_time> \mid t \in R^d,$
$arrival_time$은 t의 도착시간$\}$

이제 데이터 스트림 S에 대한 연속 스카이라인 질의를 정의한다.

정의 1 : 두 개의 d-차원 튜플 $t = (v_1, v_2, \cdots, v_d)$, $t' = (u_1, u_2, \cdots, u_d) \in R^d$에 대해, 만약 $\forall i, v_i \leq u_i$이고 $\exists i, v_i < u_i$이기만 하면, t는 t'를 지배(dominate)한다고 한다. □

본 논문에서는 t가 t'를 지배하면 $t \prec t'$로 표시하고, 그렇지 않은 경우 $t \nprec t'$로 표시한다.

정의 2 : 주어진 튜플의 집합 T에 대해, 어떤 튜플 $t \in T$가 만약 $\forall t' \in T, t' \nprec t$이기만 하면 t는 스카이라인 튜플이라고 한다. □

정의 3 : 주어진 튜플의 집합 T에 대해, T의 스카이라인을 $Skyline(T)$로 표시하고 다음과 같이 정의한다. □

$Skyline(T) = \{t \in T \mid t$는 스카이라인 튜플$\}$

제 1장에서 언급한 바와 같이, 데이터 스트림 S에는 시간의 흐름에 따라 무한개의 튜플이 포함될 수 있다. 따라서 S에 대한 스카이라인 질의를 처리하기 위해서는 RANGE와 SLIDE가 명시된 슬라이딩 윈도우가 정의되어야 한다[1]. 어떤 시간 구간$[s, e]$ 내에 도착한 튜플들을 $T([s, e])$라고 표시하자. 즉, $T([s, e]) = \{t \mid <t, arrival_time> \in S \wedge (s \leq arrival_time \leq e)\}$이다. 어떤 스카이라인 질의 q에 대한 슬라이딩 윈도우의 RANGE를 $q.R$이라 표시하고, SLIDE를 $q.S$로 표시하면, q는 매 $q.S$초마다 다음과 같은 스카이라인 결과를 출력한다.

$Skyline(T([t_{current} - q.R, t_{current}]))$

여기서 $t_{current}$는 스카이라인 결과를 출력해야 하는 현재 시간을 나타낸다. 즉, q는 매 $q.S$초마다 최근 $q.R$시간 내에 도착한 튜플들에 대한 스카이라인을 출력한다. 여기서 q가 스카이라인 결과를 출력해 주어야 하는 각 구간 $[t_{current} - q.R, t_{current}]$을 q의 질의 윈도우(query window)라고 한다.

본 논문에서는 하나 이상의 연속 스카이라인 질의가 수행되는 경우를 고려한다. n개의 연속 스카이라인 질의 $q_1, q_2, \cdots, q_n$이 주어지고 각 q_i에 대한 슬라이딩 윈도우의 RANGE와 SLIDE가 각각 $q_i.R$와 $q_i.S$라고 하자. 다중 연속 스카이라인 질의에서는 각 q_i가 매 $q_i.S$초마다 $Skyline(T([t_{current} - q.R, t_{current}]))$를 출력한다. 본 논문에서는 이러한 다중 연속 스카이라인 질의를 효율적으로 처리하는 방법을 제안한다.

3.2 단순 처리 방법

본 절에서는 다중 연속 스카이라인 질의를 처리하기 위한 가장 단순한 방법으로서 SQO

(Single Query Optimization)과 FSS (Fragment-based Skyline Sharing)를 소개한다.

3.2.1 SQO(Single Query Optimization)

다중 연속 스카이라인 질의를 처리하는 가장 쉬운 방법은 각각의 질의를 독립적으로 처리하는 것이다. 각 질의의 처리를 위해서는 [14]와 같이 단일 연속 스카이라인 질의 처리를 위해 제안된 방법을 사용하면 된다. 단, [14]에서 제안한 방법은 SLIDE = 1인 슬라이딩 윈도우만을 고려하고 있기 때문에, 이를 SLIDE = s(s = 1, 2, …)인 일반적인 경우에도 사용 가능하도록 확장해야 한다.

n개의 질의 q_1, q_2, …, q_n이 주어지면 SQO는 각 qi(i = 1, 2, …, n)에 대해 $DB_{sky}{}^i$와 $DB_{rest}{}^i$를 할당한다. 데이터 스트림 S에 새 튜플 t_{new}가 도착하면 각 q_i에 대해 다음의 작업을 수행한다.

① $DB_{sky}{}^i$에 t_{new}를 지배하는 튜플이 있는지 확인한다.
② 만약 그러한 튜플이 있다면 t_{new}를 $DB_{rest}{}^i$에 삽입하고 종료한다.
③ 만약 그러한 튜플이 없다면 $DB_{sky}{}^i$에 t_{new}가 지배하는 튜플이 있는지 확인하고, 해당 튜플을 $DB_{sky}{}^i$로부터 제거한다.
④ t_{new}를 $DB_{sky}{}^i$에 삽입한다.

또한, 각 q_i에 대해서는 매 $q_i.S$ 초마다 다음의 작업을 수행한다.

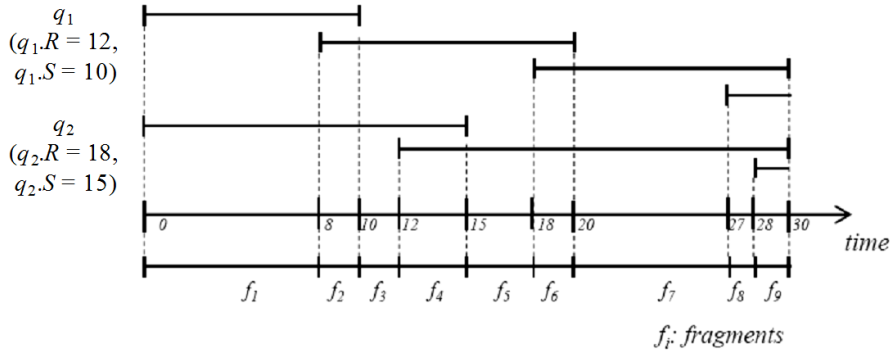
① $DB_{sky}{}^i$에서 만료된 튜플을 제거한다.
② $DB_{rest}{}^i$로부터 상기 만료된 튜플에 의해서만 지배되던 튜플을 찾아 이를 $DB_{sky}{}^i$에 삽입한다.
③ $DB_{sky}{}^i$에 포함된 튜플들을 q_i의 스카이라인 질의 결과로 출력한다.

이 방법은 간단하지만, 각각의 질의를 독립적으로 처리하기 때문에 질의 간의 계산 결과가 전혀 재사용되지 않는 단점이 있다. 따라서, 질의 수가 증가할수록 효율성이 크게 떨어지게 된다.

3.2.2 FSS(Fragment-based Skyline Sharing)

SQO 방법의 단점을 극복하면서 다중 연속 스카이라인 질의를 처리하는 다른 간단한 방법은 질의 윈도우들을 조각(*fragment*)들로 분할한 뒤, 각 조각에 대한 스카이라인을 공유하는 것이다. <그림 3>은 두 스카이라인 질의 q_1과 q_2의 질의 윈도우들을 조각으로 나누는 예를 보여준다. <그림 3>에서 $q_1.R$ = 12, $q_1.S$ = 10이고 $q_2.R$ = 18, $q_2.S$ = 15이다. 주어진 슬라이딩 윈도우의 정의에 따라, q_1는 [0, 10], [8, 20], [18, 30], …의 질의 윈도우 내에 도착한 튜플들에 대한 스카이라인을 각각 출력하고, q_2는 [0, 15], [12, 30], …의 질의 윈도우 내에 도착한 튜플들에 대한 스카이라인을 각각 출력한다. FSS 방법은 q_1과 q_2 간의 중간 결과를 공유하기 위해서 <그림 3>에서와 같이 q_1과 q_2의 질의 윈도우들을 조각 f_1, f_2, …, f_9으로 분할한다. 질의 윈도우들을 조각으로 나누는 방법은 질의 윈도우들의 시작점과 끝점들을 모아 시간 순으로 정렬한 결과를 p_1, p_2, … p_m이라고 했을 때, [p_1, p_2], [p_2, p_3], …, [p_{m-1}, p_m]이 각각의 조각이 된다.

FSS 방법은 각 조각에 대해 해당 조각 내



〈그림 3〉 질의 윈도우들을 조각(fragment)으로 나누는 예

에 도착한 튜플들에 대한 스카이라인을 계산해 놓는다. 현재 시간 $t_{current}$가 어떤 질의 윈도우의 끝점과 같으면, 해당 질의 윈도우에 대한 스카이라인을 출력해야 한다. 이를 위해 FSS는 해당 질의 윈도우에 포함된 조각들을 모두 찾아, 각 조각들에 대해 계산된 스카이라인들을 모아서 이로부터 최종 스카이라인을 구한다. 즉, 어떤 질의 윈도우 w가 i개의 조각 $f_1, f_2, \cdots, f_i$로 분할되었을 때, 즉 $w = f_1 \cup f_1 \cup \cdots \cup f_i$일 때, w에 대한 스카이라인 $Skyline(T(w))$는 다음과 같이 계산된다.

$$Skyline(T(w)) = Skyline(Skyline(T(f_1)) \cup Skyline(T(f_2)) \cup \cdots \cup Skyline(T(f_i)))$$

이 방법은 각 조각에 대한 스카이라인을 여러 질의가 공유하므로, 각각의 질의를 독립적으로 처리하는 SQO 방법에 비해서 좋은 성능을 보일 수 있다. 하지만 이 방법은 각 조각마다 스카이라인을 계산해 두어야 하며, 한 질의 윈도우에 대한 스카이라인을 출력하기 위해서는 해당 질의 윈도우에 포함된 조각들에 대한 스카이라인들을 모아 최종 스카이라인을 계산해야 하는 비용이 든다. 따라서 조각의 수가 증가할수록 질의 처리 비용이 커지게 된다.

4. 제안 방법

본 장에서는 다중 연속 스카이라인 질의 처리를 위해 본 논문에서 제안하는 방법을 설명한다. 제안하는 방법은 제 3.2.2절에서 설명한 FSS 방법과 동일하게 질의 윈도우들을 조각으로 나눈다. 하지만 제안하는 방법은 FSS와 달리, 각 조각에 대해 스카이라인을 계산해 두는 대신, 각 조각에 대해 반드시 하나 이상의 질의 윈도우에 대한 스카이라인 튜플이 될 튜플들만을 가려내어 이들을 메모리에 저장해 놓는다. 또한, 메모리에 저장되는 각 튜플에 대해 해당 튜플이 어떤 질의 윈도우에 대한 스카이라인 튜플이 될지를 알려주는 정보를 같이 저장한다. 이로 인해 제안 방법은 각 질의 윈도우의 스카이라인을 매우 효율적으로 얻어낼 수 있으며, 질의 수가 증가하더라도 각 질의에 대한 결과를 매우 빠

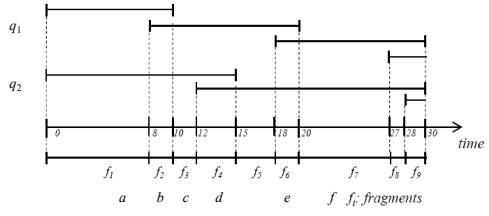
르게 출력할 수 있다.

본 장에서는 제안하는 방법을 (1) 새로운 튜플이 도착할 때마다 수행되는 작업과 (2) 어떤 질의 윈도우의 스카이라인을 출력해야 할 때 수행되는 작업으로 나누어 설명한다.

4.1 도착 튜플 처리

제안 방법은 새로운 튜플 t_{new}가 도착하면, t_{new}가 최소한 하나 이상의 질의 윈도우에 대한 스카이라인 튜플이 될 튜플일지를 판단한다. 만약 그렇다면 t_{new}를 메모리에 저장하고, 그렇지 않다면 t_{new}를 버린다. 이에 따라 메모리에는 스카이라인 튜플로서 최소한 한 번 이상은 출력될 튜플만이 저장된다. t_{new}가 최소한 하나 이상의 질의 윈도우에 대한 스카이라인 튜플이 될지를 판단하는 방법은 다음과 같다.

먼저 t_{new}가 속한 조각이 포함된 모든 질의 윈도우들 중 가장 늦은 시작점을 가지는 질의 윈도우를 찾는다. 해당 질의 윈도우를 $[s_w, e_w]$라고 하고, t_{new}가 속한 조각을 $[s_f, e_f]$라고 하자. 여기서 $[s_w, e_f]$의 구간을 고려하자. 만약 $[s_w, e_f]$ 내에 t_{new}를 지배하는 튜플 t'가 존재한다면, t_{new}는 모든 질의 윈도우에 대해서 t'와 같이 존재하므로 절대로 어떤 질의 윈도우에 대해서도 스카이라인 튜플이 될 수 없다. 반면에, $[s_w, e_f]$ 내에 t_{new}를 지배하는 튜플이 존재하지 않는다면, t_{new}는 최소한 윈도우 $[s_w, e_w]$에 대해서는 스카이라인 튜플이 될 가능성이 있음을 알 수 있다. 예를 들어, <그림 4>에서 $b < c$이고 $d < e$라고 하자. 튜플 c는 [8, 12]구간 내에 그를 지배하는 튜플 b가 있으므로 어떠한 질의 윈도우에 대해서도 스카이라인 튜플이 될 수 없다. 한편, e는 [18, 20]구간 내에 그를 지배하는 튜플이 없으므로, 질의 윈도우 [18, 30]에 대해서는 스카이라인 튜플이 될 가능성이 있다.



〈그림 4〉 스카이라인 여부 판단 구간

정의 4 : 주어진 튜플 t가 속한 조각을 $f = [s_f, e_f]$라 하고, f를 포함하는 모든 질의 윈도우들 중 가장 늦은 시작점을 가지는 윈도우를 $w = [s_w, e_w]$라고 하자. 이때 시간 구간 $[s_w, e_f]$ 를 t에 대한 '스카이라인 여부 판단 구간'이라 하고 $SDI(t)$라고 표기한다.

제안하는 방법은 새로운 튜플 t_{new}가 도착하면, $SDI(t_{new})$ 내에 t_{new}를 지배하는 튜플이 존재하는지를 보고 그의 여부에 따라 t_{new}가 스카이라인 튜플이 될 가능성을 판단한다. 만약 $SDI(t_{new})$ 내에 t_{new}를 지배하는 튜플이 존재한다면, t_{new}는 스카이라인 튜플이 될 가능성이 없으므로 버린다. 만약 $SDI(t_{new})$ 내에 t_{new}를 지배하는 튜플이 존재하지 않는다면, t_{new}는 스카이라인 튜플이 될 가능성이 있으므로 메모리에 저장한다.

스카이라인 튜플이 될 가능성이 있는 튜플을 메모리에 저장할 때, 제안 방법은 해당 튜플이 어떤 질의 윈도우에 대한 스카이라인 튜플이 될지를 빠르게 판단할 수 있도록 부

가 정보를 같이 저장한다. 먼저 주어진 튜플 t에 대해 *MinStart*(*t*)와 *MaxStart*(*t*)를 다음과 같이 정의한다.

정의 5 : 주어진 튜플 *t*에 대해, *t*가 포함된 질의 윈도우들 중 *t*를 지배하는 튜플이 없는 질의 윈도우들 중에서 가장 빠른 시작점을 가진 질의 윈도우를 [*s*, *e*]라고 하자. 이때, *MinStart*(*t*) = *s*이다.

정의 6 : 주어진 튜플 *t*에 대해 *t*의 도착 시점 이후에 가장 먼저 시작되는 질의 윈도우를 [*s*, *e*]라고 하자. 이 때, *MaxStart*(*t*) = *s*이다.

보조정리 1 : 어떤 튜플 *t*와 질의 윈도우 [*s*, *e*]에 대해, *t*는 $s \in$ [*MinStart*(*t*), *MaxStart*(*t*)]을 만족하는 질의 윈도우 [*s*, *e*]에 대해서만 스카이라인 튜플이 될 수 있다.

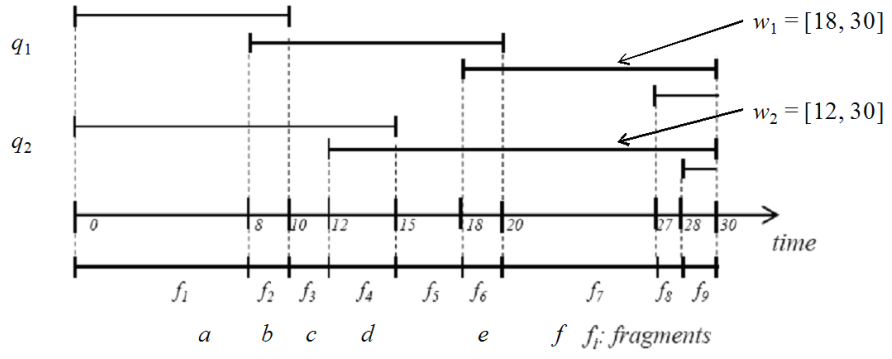
증명 : 정의 5에 따라 *t*는 시작점이 *MinStart*(*t*)보다 빠른 질의 윈도우에 대해서는 스카이라인 튜플이 될 수 없다. 또한, *t*는 시작점이 *MaxStart*(*t*)보다 늦는 질의 윈도우에 대해서는 그에 포함되지 않으므로 역시 스카이라인 튜플이 될 수 없다. 따라서 *t*는 그의 시작점이 *MinStart*(*t*)보다 늦고 *MaxStart*(*t*)보다 빠른 질의 윈도우에 대해서만 스카이라인 튜플이 될 수 있으므로 보조정리가 성립된다. □

보조정리 1에 의해 어떤 튜플 *t*는 시작점이 [*MinStart*(*t*), *MaxStart*(*t*)]사이에 존재하는 질의 윈도우에 대해서만 스카이라인 튜플이 될 수 있다는 것을 알 수 있다. 제안하는 방법은 메모리에 저장되는 각 튜플 *t*에 대해 부가적으로 *MinStart*(*t*)와 *MaxStart*(*t*)를 계산하여 저장한다. 이러한 정보는 어떤 질의 윈도우에 대한 스카이라인 결과를 출력할 때, 해당 질의 윈도우에 포함된 튜플들 중 어떤 튜플들이 스카이라인에 포함될지를 빠르게 판단하는데 사용된다. 이의 사용은 다음 절에서 자세히 설명한다.

4.2 스카이라인 출력

현재 시간 $t_{current}$가 어떤 질의 윈도우의 끝점과 같으면, 해당 질의 윈도우에 대한 스카이라인을 출력해야 한다. 앞서 설명한 바와 같이, 제안하는 방법은 (1) 적어도 하나 이상의 질의 윈도우의 스카이라인 튜플이 될 튜플만을 메모리에 저장하며, (2) 메모리에 저장되는 각 튜플 *t*에 대해서는 어떤 질의 윈도우의 스카이라인 튜플이 되는지 판단하는 데 사용되는 정보인 *MinStart*(*t*)와 *MaxStart*(*t*)를 같이 저장한다. 따라서 제안하는 방법은 질의 윈도우가 여러 개일 때도 각각에 대한 스카이라인을 매우 빠르게 얻어낼 수 있다.

현재 시간 $t_{current}$에 스카이라인을 출력해야 하는 질의 윈도우를 $w_1 = [s_1, t_{current}]$, $w_2 = [s_2, t_{current}]$, …, $w_m = [s_m, t_{current}]$라고 하자. <그림 5>는 두 개의 질의 윈도우 w_1 = [18, 30]과 w_2 = [12, 30]에 대한 스카이라인 *Skyline*($T(w_1)$)과 *Skyline*($T(w_2)$)를 출력하는 과정을 나타낸다. 제안하는 방법은 가장 최근 조각부터 w_1, w_2, …, w_m이 포함하고 있는 가장 오래된 조각까지



〈그림 5〉 두 질의 윈도우에 대한 스카이라인 출력 예

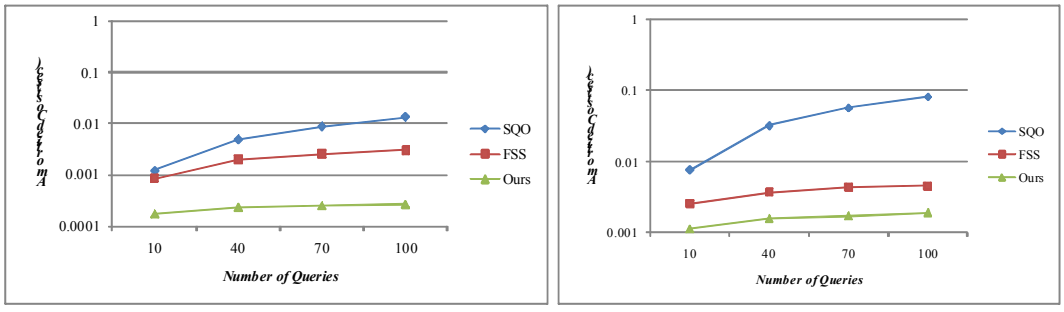
진행한다. <그림 5>에서는 w_1과 w_2가 포함하고 있는 조각들이 { f_4, f_5, f_6, f_7, f_8, f_9}이므로 $f_9 \rightarrow f_8 \rightarrow \cdots \rightarrow f_4$ 순으로 진행하게 된다.

제안하는 방법은 현재 조각 내에 속한 튜플들 중 메모리에 저장된 튜플들에 대해, 각 튜플 t의 *MinStart*(t)와 *MaxStart*(t)를 보고 각 질의 윈도우 $w_i = [s_i, t_{current}]$ (i = 1, 2, ⋯, m)에 대해 $s_i \in$ [*MinStart*(t), *MaxStart*(t)]인지를 검사한다. 만약 $s_i \in$ [*MinStart*(t), *MaxStart*(t)]라면 t를 질의 윈도우 $w_i = [s_i, t_{current}]$의 스카이라인

```
Procedure OnTupleArrival()
t_new = a newly arrived tuple ;
If (there exists a tuple in SDI(t_new) that dominates t_new)
    drop t_new ;
Else
    store < t_new, MinStart(t_new), MaxStart(t_new) > in memory ;
End procedure

Procedure OutputSkylines
w1, w2, ⋯ wm = query windows whose end point is t_current;
f_oldest = the oldest fragment of w1, w2, ⋯ wm ;
f_newest = the most recent fragment ;
For (f = f_newest to f_oldest) {
    For (each tuple t in f that is stored in memory) {
        For (each wi = [si, t_current] (i = 1, 2, ⋯, m)) {
            If (MinStart(t) <= si <= MaxStart(t))
                output t as a skyline tuple of wi ;
        }
    }
}
End procedure
```

〈그림 6〉 제안 방법을 나타내는 알고리즘



(a) Independent (b) Anti-correlated

〈그림 7〉 질의 수 변화에 따른 성능

튜플로 출력한다. 이와 같은 절차를 w_1, w_2, …, w_m이 포함하고 있는 가장 오래된 조각까지 수행한다. <그림 5>에서는 f_4까지 수행하게 된다. 따라서, 제안하는 방법은 w_1, w_2, …, w_m이 포함하고 있는 조각들에 속한 튜플들을 한번만 스캔하고서도 주어진 m개의 질의 윈도우 w_1, w_2, …, w_m의 스카이라인을 동시에 빠르게 내보낼 수 있다.

<그림 6>은 지금까지 설명한 제안 방법을 나타내는 알고리즘이다. OnTupleArrival( )는 새로운 튜플이 도착했을 때마다 수행되는 알고리즘이며, OutputSkylines( )는 현재 시간 $t_{current}$가 어떤 질의 윈도우의 끝점과 같을 때 수행되는 알고리즘이다.

5. 성능 평가

본 장에서는 제안하는 방법의 성능을 실험을 통해 평가한다. 실험에서는 제안하는 방법을 제 3.2절에서 기술한 SQO 및 FSS와 비교하였다. 각 알고리즘은 C++ 언어로 구현하였으며, 실험은 Pentium IV 2.5GHz CPU를 가진 PC에서 수행되었다. 실험 데이터로는 [3]에서 사용된 합성 데이터 생성기를 사용하여 다차원 데이터를 생성하였다. 합성 데이터는 d-차원 튜플들로 이루어져 있으며, 각 차원에 대한 값은 0에서 1사이의 값을 갖는다. 차원들 간의 값의 분포는 스카이라인 연구에서 흔히 사용되는 분포로서 (1)독립(independent)과 (2)반상관(anti-correlated) 분포를 사용하였다. 실험에서는 주어진 개수의 질의들을 총 10,000초 동안 수행하여 튜플 당 처리 시간을 측정하였다. 각 실험에서 튜플의 도착 속도는 10개/second로 설정하였으며, 튜플의 차원은 2차원에서 5차원까지 변화시켰다. 각 질의의 슬라이딩 윈도우는 RANGE는 [600, 900]사이의 값에서 SLIDE는 [300, 600]사이의 값에서 무작위로 추출하여 설정하였다.

5.1 질의 수 변화에 따른 성능

첫 번째 실험에서는 동시에 수행되는 질의의 수를 변화시켜 가면서 세 가지 방법의 성능을 측정하였다. <그림 7>은 질의의 수를 10, 40, 70, 100으로 증가시켰을 때, 각 방법에

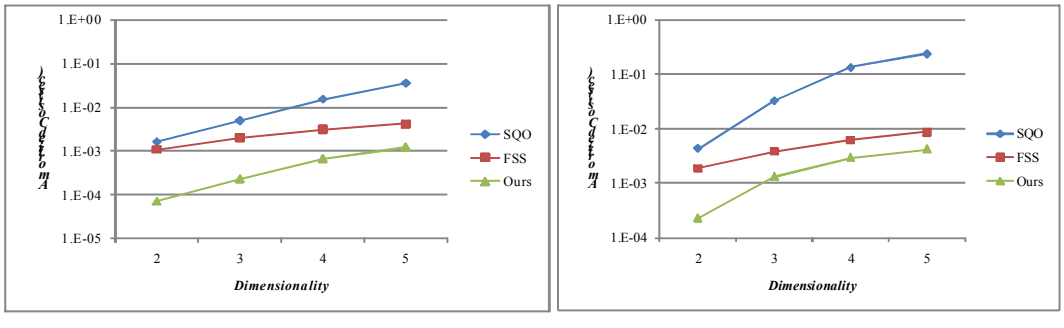
서의 튜플 당 처리 시간을 나타낸 것이다. <그림 7(a)>와 <그림 7(b)>는 각각 차원들 간의 값의 분포가 독립일 때와 반상관일 때를 나타낸다. 그림에서 볼 수 있듯이 FSS 방법은 각 질의를 독립적으로 처리하는 SQO 방법보다는 좋은 성능을 보이고 있다. 또한 제안하는 방법은 모든 경우에 대해서 SQO와 FSS보다 최소 10배 이상의 성능을 보이고 있음을 알 수 있다. 특히 제안하는 방법은 질의 수가 증가함에 따라서 다른 방법보다 성능이 천천히 감소하고 있다. 따라서 제안하는 방법은 질의 수 증가에 대해 다른 방법에 비해 더 좋은 확장성을 가지고 있음을 알 수 있다.

5.2 차원 수 변화에 따른 성능

다음 실험에서는 튜플의 차원 수를 2에서 5까지 증가시켜가면서 세 가지 방법의 성능을 측정하였다. 본 실험에서 질의 수는 40개로 고정하였다. <그림 8>은 차원 수 변화에 따른 실험 결과를 나타낸다. <그림 8(a)>와 <그림 8(b)>는 각각 차원들 간의 값의 분포가 독립일 때와 반상관일 때를 나타낸다. 그림에서 볼 수 있듯이, 제 4.1절에서의 실험과 마찬가지로 FSS는 SQO보다 좋은 성능을 보이고 있으며, 제안하는 방법은 모든 경우에 대해 두 방법보다 더 좋은 성능을 보이고 있다.

6. 결 론

e-Business에서 데이터 스트림의 처리는 점차 중요성이 커지고 있는 연구 주제 중의 하나이다. 그 중 데이터 스트림에 대한 연속 스카이라인 질의는 여러 응용에서 사용되고 있지만, 아직까지 다중 연속 스카이라인 질의에 대해서는 거의 연구가 이루어지지 않았다. 본 논문은 데이터 스트림 환경에서 다중 연속 스카이라인 질의 처리를 위한 효율적인 방법을 제안하였다. 제안하는 방법은 주어진 질의 윈도우들을 조각으로 나누고, 각 조각에 대해 적어도 하나 이상의 질의 윈도우의 스카이라인 튜플로 출력될 튜플만을 메모리에 저장한다. 또한 부가적인 정보를 사용하여, 메모리



(a) Independent (b) Anti-correlated

〈그림 8〉 차원 수 변화에 따른 성능

에 저장된 각 튜플들이 어떤 질의 윈도우의 스카이라인 튜플로 출력될 튜플인지를 효율적으로 알아낸다. 이러한 방법을 통해 제안하는 방법은 여러 개의 연속 스카이라인 질의가 수행되는 환경에서도 각각의 질의에 대한 결과를 매우 빠르게 내보낼 수 있다. 실험을 통해 제안하는 방법은 각 질의를 독립적으로 수행하는 방법과 단순 공유 방법에 비해 10배 이상의 성능 향상을 보임을 보였다.

참 고 문 헌

[1] Arasu, A., Babu, S., and Widom, J., "The CQL continuous query language : Semantic foundations and query execution," The VLDB Journal, Vol. 15, No. 2, 2006, pp. 121-142.

[2] Bentley, J. L., Kung, H. T., Schkolnick M., and Thompson, C. D., "On the average number of maxima in a set of vectors and applications," Journal of the ACM, Vol. 25, No. 4, 1978, pp. 536-543.

[3] Borzonyi, S., Stocker, K., and Kossmann, D., "The skyline operator," Proceedings of the 17th International Conference on Data Engineering, 2001.

[4] Chomicki, J., Godfrey, P., Gryz, J., and Liang, D., "Skyline with presorting," Proceedings of the 19th International Conference on Data Engineering, 2003.

[5] Godfrey, P., Shipley, R., and Gryz, J., "Maximal vector computation in large data sets," Proceedings of the 31st international conference on Very Large Data Bases, 2005.

[6] Huang, Z., Sun, S., and Wang, W., "Efficient mining of skyline objects in subspaces over data streams," Knowledge and Information Systems, Vol. 22, No. 2, 2010, pp. 159-183.

[7] Kossmann, D., Ramsak, F., and Rost, S., "Shooting stars in the sky : An online algorithm for skyline queries," Proceedings of the 28th international conference on Very Large Data Bases, 2002.

[8] Kung, H. T., Luccio, F., and Preparata, F. P., "On finding maxima of a set of vectors," Journal of the ACM, Vol. 22, No. 4, 1975, pp. 469-476.

[9] Lee, K. C., Lee, W., Zheng, B., Li, H., and Tian, Y., "Z-SKY : an efficient skyline query processing framework based on Z-order," The VLDB Journal, Vol. 19, No. 3, 2010, pp. 333-362.

[10] Lin, X., Yuan, Y., Wang, W., Lu, H., "Stabbing the sky : Efficient skyline computation over sliding windows," Proceedings of the 21st International Conference on Data Engineering, 2005.

[11] Papadias, D., Tao, Y., Fu, G., Seeger, B., "An optimal and progressive algorithm for skyline queries," Proceedings of the 2003 ACMSIGMOD international conference on Management of data, 2003.

[12] Sun, S., Huang, Z., Zhong, H., Dai, D., Liu, H., and Li, J., "Efficient monitoring of skyline queries over distributed data streams," Knowledge and Information Systems, DO I : 10.1007/s10115-009-0269-0, 2009.

[13] Tan, K. L., Eng, P. K., and Ooi, B. C., "Efficient progressive skyline computa-

tion," Proceedings of the 27th International Conference on Very Large Data Bases, 2001.

[14] Tao, Y. and Papadias, D., "Maintaining sliding window skylines on data streams," IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 3, 2006, pp. 377-391.

저 자 소 개

이유원 (E-mail : ywlee@dbserver.kaist.ac.kr)
2005년 부산대학교 전자전기정보컴퓨터공학부 정보컴퓨터공학전공 (학사)
2007년 한국과학기술원 전산학과 (석사)
2007년~현재 한국과학기술원 전산학과 박사과정
관심분야 데이터베이스 시스템, 센서 네트워크, 스트림 데이타 처리 등

이기용 (E-mail : kiyonglee@sookmyung.ac.kr)
1998년 KAIST 전산학과 (학사)
2000년 KAIST 전산학과 (석사)
2006년 KAIST 전자전산학과 전산학전공 (박사)
2006년~2008년 삼성전자 기술총괄 소프트웨어연구소 책임연구원
2008년~2010년 KAIST 전산학전공 연구조교수
2010년~현재 숙명여자대학교 컴퓨터과학과 조교수
관심분야 정보시스템, 소프트웨어 시스템, 질의 처리

김명호 (E-mail : mhkim@dbserver.kaist.ac.kr)
1982년 서울대학교 컴퓨터 공학과 (학사)
1984년 서울대학교 컴퓨터 공학과 (석사)
1989년 Michigan State Univ. 전산학 (박사)
1989년~현재 한국과학기술원 전산학전공 교수
1995년 Univ. of Virginia 방문 교수
관심분야 Database, Distributed Systems, Workflow, Multimedia, OLAP, Data Warehouse