GAGPC: 데이타 스트림에 대한 다중 연속 질의의 최적화 알고리즘

(GAGPC: An Algorithm to Optimize Multiple Continuous Queries on Data Streams)

서 영 균 † 손 진 현 †† 김 명 호 †††
(Young-Kyoon Suh) (Jin Hyun Son) (Myoung Ho Kim)

요 약 데이타 스트림에 대한 다중 연속 질의들 사이에는 질의들의 윈도우 중첩 및 주기적 실행 간격으로 인해 재사용이 가능한 중간 결과들이 다수 생길 수 있다. 본 논문은 다중 연속 질의들을 위한 전체 실행 계획을 구성하기 위해, 효율적인 탐욕 기반의 경험적 알고리즘인 GAGPC를 제안한다. 제안한 GAGPC 알고리즘은 질의들의 전체 실행 사이클을 결정하고 관련된 실행 시점들의 최대 집합인 SRP를 찾는다. 다음, 각 SRP에서 실행될 질의들이 가장 높은 이익을 갖는 공통의 조인 부분들을 공유하도록 전체 실행 계획을 구성한다. 본 논문은 공통된 질의 부분의 존재뿐만 아니라 그것과 관련된 중첩된 윈도우 크기에 따라 동일한 연속 질의라 하더라도 최상의 질의 계획이 바뀔 수 있다는 점을 제시한다. 또한 기존 연구와는 달리, 윈도우가 부분 또는 전체적으로 중첩될 수 있으므로 중간 결과의 전체뿐만 아니라 일부도 재사용할 것을 반영한다. 마지막으로, 본 논문은 GAGPC의 유효성을 위한 시뮬레이션 결과를 제시한다.

키워드 : 다중 연속 질의, 데이타 스트림, 센서 네트워크

Abstract In general, there can be many reusable intermediate results due to the overlapped windows and periodic execution intervals among Multiple Continuous Queries (MCQ) on data streams. In this regard, we propose an efficient greedy algorithm for a global query plan construction, called GAGPC. GAGPC first decides an execution cycle and finds the maximal Set(s) of Related execution Points (SRP). Next, GAGPC constructs a global execution plan to make MCQ share common join-fragments with the highest benefit in each SRP. The algorithm suggests that the best plan of the same continuous queries may be different according to not only the existence of common expressions, but the size of overlapped windows related to them. It also reflects to reuse not only the whole but partial intermediate results unlike previous work. Finally, we show experimental results for the validation of GAGPC.

Key words : Multiple Continuous Queries (MCQ), Data Streams, Sensor Network

1. 서 론

기존 데이타베이스는 영속적인 저장소에 저장된 관련 데이타 집합을 의미한다. 반면에, 최근에 많은 관심과 연구가 수행되고 있는 데이타 스트림(data stream)은 정보 소스로부터 끊임없이 생성되는 데이타의 연속(sequence of data)을 의미하며, 기존의 데이타베이스 환경과는 다른 데이타 관리 기법을 요구한다[1]. 센서 데이타[2-4], 주식 데이타[5], 위치 기반 데이타 등이 데이타 스트림의 대표적인 예가 된다. 데이타 스트림은 물이 흐르는 것과 같이 끊임없이 생성되고, 스트림의 각 데이타는 크기는 상대적으로 작으나, 그 양이 무한하고 가변적인 특성을 갖는다. 이러한 특징으로 인해 사용자는 새로운 데이타가 도착함에 따라 계속적으로 새로운 결과를 요구하게 되어, 한 번 요청된 질의가 계속적으로 실행될 수 있어야 한다. 이 질의를 일반적으로 연속 질의(continuous query)라고 부른다[6,7].

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성지원 사업(IITA-2005-C1090-0502-0016)의 연구결과로 수행되었으며, 또한 한국과학재단 목적기초연구(R08-2003-000-10464-0) 지원으로 수행되었음

† 정 회 원 : 한국과학기술정보연구원 NTIS 사업단 통합기술개발팀 연구원
yksuh@kisti.re.kr
†† 종신회원 : 한양대학교 컴퓨터공학과 교수
jhson@cse.hanyang.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr

논문접수 : 2005년 9월 9일
심사완료 : 2006년 3월 26일

데이타 집합(stored set)에 전달되어 한번만 실행되는 애드-혹(ad-hoc) 질의와는 달리 연속 질의[6, 7]에는 윈도우(window)와 실행 간격(execution interval)이 함께 주어진다. 기존 데이타베이스에서 조인 연산(join operation)은 이미 지정된 테이블에 속한 한정된 데이타에 대해 수행된다. 그러나, 데이타 스트림에서는 조인 연산의 대상이 되는 어느 한쪽의 데이타 스트림을 다 읽지 않고서는 기존 데이타베이스에서와 같은 조인 연산을 수행하는 것이 불가능하다. 최근의 데이타 스트림 연구에서는 '윈도우'[8]라는 개념을 이용하여 조인 연산의 대상인 스트림을 제한하고 있다. 윈도우는 '최근 t 시간까지 도착한 데이타' 또는 '최근에 도착한 T 개의 데이타' 등으로 정의된다. 그림 1은 윈도우 내에 존재하는 스트림 A와 스트림 B의 데이타에 대해 윈도우 조인(window join) 연산을 수행하는 과정을 보여준다. 또한 '실행 간격'[5]은 연속 질의를 얼마나 주기적으로 실행할 것인지를 나타낸다. 예를 들어 "10분마다 최근 2분 내에 도착한 스트림 A의 *loc* 값과 스트림 B의 *loc* 값이 동일할 때의 스트림 A의 *velocity*를 알려 주시오" 라는 연속 질의에서 실행 간격은 '10분'이 된다.

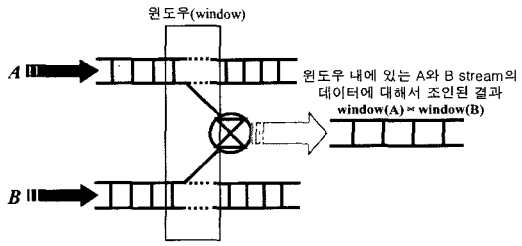

그림 1 스트림 A과 스트림 B의 데이타에 대한 윈도우 조인 연산

윈도우와 실행 간격이라는 새로운 연산자로 인해 기존 데이타베이스 연구에서 논의되어 온 질의 최적화 알고리즘을 있는 그대로 적용하기 어렵다. 기존 데이타베이스의 다중-질의-최적화(multiple-query-optimization) 알고리즘은 공통된 하위-표현에 의해 생성되는 중간 결과들을 재사용할 것을 고려하는 공유 계획 기법을 이용한다. 그러나, 데이타 스트림에서는 이러한 관점 외에도 다음의 세 가지 사항들이 추가적으로 논의되어야 한다. 첫째, 다중 연속 질의들은 서로 다른 크기의 윈도우 내에 들어온 데이타에 대한 조인을 요구하면서 서로 다른 간격으로 실행되므로, 공통된 질의 부분에 대한 중간 결과 전체에 대해서 보다는 부분적으로 공유가 가능한 경우가 많이 발생하게 된다. 둘째, 연속 질의 간에 공유 가능한 중간 결과의 크기에 따라 중간 결과를 재사용하도록 하는 공유 계획이 효과적일 수도 있고 그렇지 않을 수도 있다. 기존 연구의 알고리즘은 질의 간에 공통이 되는 전체의 중간 결과만을 공유할 것을 고려하므로, 부분적으로 공유 가능한 중간 결과도 고려할 수 있도록 수정되어야 한다. 마지막으로, 동일한 연속 질의가 실행되더라도 다른 연속 질의가 생성한 전체 또는 일부 중간 결과의 재사용 여부에 따라, 그 질의의 최상의 실행 계획은 매 실행 시점마다 변경될 수 있다.

본 논문에서는 데이타 스트림 환경에서 부가적으로 고려되어야 하는 위의 세 가지 특징들을 기반으로, 다중 연속 질의들에 대한 실행 계획(execution plan)을 세우는 탐욕 스타일의 경험적 기법(greedy-style heuristic approach)을 제안한다. 본 논문에서 제안하는 기법은 윈도우 크기와 실행 간격을 사용하여 정의되는 재사용 비율(reusable ratio)과 이익(benefit) 모델을 기반으로 한다. 기존 연구와는 달리, 제안하는 방법은 달리 다른 질의가 생성한 중간 결과의 일부를 재사용하는 공유 계획까지 고려함으로써 전체적으로 더 효율적인 실행 계획을 세울 수 있게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해서 언급하고, 3장에서는 본 논문에서 다루는 문제가 정의된다. 4장에서 문제를 해결하는 개략적인 아이디어를 설명한 다음, 5장에서 본 논문의 알고리즘을 제시한다. 6장에서는 제안한 알고리즘의 동작 예제를 보이고, 시뮬레이션을 통한 실험 결과를 바탕으로, 제안한 기법의 유효성을 보인다. 마지막 7장에서는 결론을 맺고 추후에 연구해야 할 방향에 대해서 언급한다.

2. 관련 연구

본 논문에서 논의하고 있는 다중 질의 최적화는 기존의 관계형(relational) 데이타베이스 질의 처리 분야[9]에서 많은 연구가 시작되었다. 그림 2와 같이 여러 질의 사이에 공통 표현이 존재할 때, 전체 질의의 실행 비용을 최소화시킬 수 있도록 중간 결과를 공유하는 실행 계획을 세우는 것이다. 그러나, 데이타 스트림 환경에서 고려해야 할 윈도우 크기와 실행 간격에 대해서는 논의하지 않았다.

데이타 스트림과 XML 분야에서도 최근 몇 년간 본 논문에서 고려하고 있는 문제를 심도 있게 다루고 있다.

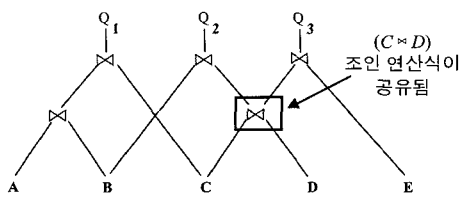

그림 2 저장된 데이타 집합에서의 다중 질의 최적화

[3]에서는 'eddy'라는 질의 처리 체제를 기반으로 데이타 스트림에서 지속적으로 환경에 적응하는 연속 질의 시스템(Continuously Adaptive Continuous Queries: CACQ)을 제시하고 있다. 구체적으로, 시스템에 도착하는 튜플들이 적용되는 연산자들의 순서를 처리 상황에 따라 효율적으로 질의를 처리할 수 있는 유연한 튜플 라우팅 기법을 제안한다. 또한 비슷한 선택 술어(selection predicate)을 갖는 질의들을 그룹 필터(grouped filter)라는 술어 색인 연산자를 이용하여 그룹핑하여 처리하는 기법을 제안한다. 그러나, [3]은 본 논문에서 언급한 실행 간격 기반의 질의를 지원하지 않았다. 본 논문이 대상으로 하는 환경에 적용하기 위해, [3]의 질의에 실행 간격 및 조인을 위한 윈도우 연산자를 추가할 필요가 있다.

한편, [5]에서는 NiagaraCQ에 대해서 언급하고 있다. 많은 질의들이 비슷한 구조를 공유한다는 사실에 기반하여 연속 질의를 점증적으로 그룹핑 한다. 또한 새로운 데이타가 도착했을 때 실행되는 질의(arrival-based query)나 특정 실행 간격을 지니는 질의(interval-based query) 등의 두 가지 경우 모두를 지원한다. 그러나, [5]는 XML 문서 스트림에 제한된 다중 연속 질의 최적화 문제를 다루고 있으며, [5]에서 논의되고 있는 질의는 두 개의 스트림을 조인하기 위한 윈도우 연산자를 지원하지 않는다. 본 논문이 대상으로 하는 환경에 적용하기 위해, 질의가 대상으로 하는 스트림을 XML 문서가 아닌 일반 데이타 스트림으로 확장하고, 스트림에 대한 조인을 지원하기 위해 윈도우 연산자를 추가할 필요가 있다.

센서 네트워크(Sensor Network) 분야를 기반으로 데이타 스트림에서의 질의 처리에 관한 몇몇 다른 연구들도 존재한다[2,4]. 센서 네트워크 분야의 연구들은 값비싼 센서들의 전력 소모(power consumption)가 최소화되도록 하는 질의 처리 알고리즘을 중점적으로 다루고 있다. 본 논문에서는 전력이 항상 공급되는 데이타 스트림 서버에서 실행되는 알고리즘을 설계하므로, 전력이라는 요소를 고려하지 않았다. 그러나, 논문의 알고리즘이 전력과는 상관없이 윈도우와 실행 간격을 바탕으로 연속 질의들의 실행을 최적화하기 때문에, 센서 네트워크에 적용하더라도 크게 영향을 미치지 않을 것으로 기대한다.

3. 문제 정의

이 장에서는, 먼저 문제의 환경을 설정한 이후에 연속 질의와 윈도우 조인에 대해서 간략히 설명하고 이를 기반으로 본 논문의 문제를 정의한다.

표 1 표기법

λ	시간 단위(time unit) 당 데이터 스트림의 도착율(arrival rate)
T	시간 단위의 크기(size)
CQ	연속 질의
$A_{[t1, t2]}$	$t1$~ $t2$ 사이에 도착한 데이터 스트림 A
$\square_{[t1, t2]}$	$t1$~ $t2$ 사이에 도착한 데이터 스트림을 표현하는 단일 연산자(unary operator)

3.2 문제 정의

그림 4는 서로 다르게 설정된 윈도우와 실행 간격을 갖는 n 개의 연속 질의가 시스템에서 실행되는 것을 보여주고 있다. 이러한 환경에서 본 논문에서 논의하고자 하는 것은 n 개의 연속 질의를 실행하는 가장 효율적인 전체 계획(global plan)을 구성하는 것이다. [10]에서는 한 번 공유하기로 한 실행 계획을 바꾸지 않고 계속해서 실행하기 때문에 각 연속 질의는 최상의 계획으로 실행되지 않을 수 있는 문제점이 있지만, 본 논문에서는 동일한 질의에 대해 최상의 실행 계획이 상황에 따라 바뀔 수 있다는 것을 고려하고 있다.

전체 실행 계획을 구성하기 위해서 아래 사항들이 고려되어야 한다. 그림 4에서 관찰할 수 있듯이, n 개의 연속 질의 간에 많은 윈도우가 겹쳐져 있음을 알 수 있다. 이것은 다른 질의가 생성해 놓은 중간 결과를 재사용할 수 있는 가능성이 높다는 것을 의미한다. 한 예로, CQ_1의 조인 표현이 $A \bowtie B \bowtie C$, CQ_2의 조인 표현이 $B \bowtie C \bowtie D$, CQ_3의 조인 표현이 $C \bowtie D \bowtie E$와 같이 주어졌다고 하면, CQ_1과 CQ_2, CQ_2와 CQ_3은 서로 공통인 하위 표현을 가지므로 중간 결과를 서로 공유할 수 있는 계획을 세울 수도 있으나, CQ_1 과 CQ_3 은 공통인 하위 표현이 존재하지 않는다. 따라서 연속 질의 간의 공통 하위 표현의 존재 여부를 고려하여 전체 계획을 세워야 한다.

정의 1. 연속 질의가 가질 수 있는 자체 후보 계획 중에 최소 비용으로 실행할 수 있는 질의 계획을 **지역 질의-처리 계획**(Local Query-processing Plan)이라고 정의한다.

정의 2. 연속 질의가 가질 수 있는 자체 후보 계획 중에 다른 연속 질의와 중간 결과를 공유할 수 있도록 조정된 질의 실행 계획을 **조정 질의-처리 계획**(Adjusted Query-processing Plan for reusing) 이라고 정의한다.

또한, 매 시점의 실행 계획을 결정할 때 지역 질의-처리 계획을 세우는 것이 좋은지 아니면 조정 질의-처리 계획을 세우는 것이 좋은지를 분석해야 한다. 고려해야 할 점은 데이타 스트림에서는 기존 데이타베이스와는 달리 질의가 설정한 윈도우에 따라 전체가 아닌 일부의 중간 결과 크기가 질의 계획 선택에 결정 요소가

3.1 환경 설정

항목	환경 설정(가정) 내용 근거 또는 추가 설명
데이타 스트림	논리적 관계형 테이블(logical relation, or table)로서 정의된다. 즉, 특정 스키마를 갖는 관계형 튜플(relational tuple)들이 시간에 따라 정보 소스에서 생성되어 포아송 프로세스(Poisson Process)로 도착한다고 가정한다. 만약, 포아송 프로세스로 도착하지 않는다면 질의 처리 이외에 고려될 요소들이 증가하여 본 논문의 알고리즘이 매우 복잡해 질 수 있다.
연속 질의	하나의 연속 질의 CQ_i 는 다음과 같이, 질의 식별자(i), 조인 연산식(JE), 윈도우 크기(W), 그리고 실행 간격(T)로 구성된다. 또한 윈도우 크기는 '최근 T 시간 내에 도착한 튜플 개수'로 정의된다. CQ_i | (*ID i, Join Expression JE, Window_Size W, Interval T*) 본 논문에서는 일관성을 해치지 않는 범위에서 조인 처리 만을 고려한다. 이때 조인 처리만을 고려하는 것은 기존 데이타베이스 환경에서와 같이, 데이타 스트림에서도 의미 있는 결과를 사용자에게 전달하기 위해 그것의 효율적인 처리 방법이 매우 중요하기 때문이다.
비용 모델	스트림 환경에서는 무한한 스트림을 모두 디스크에 저장할 수 없으므로, 연속 질의 처리를 위해 데이타를 디스크로부터 읽어 들이는 것이 현실적으로 어렵다. 그래서 [8]에서의 가정과 같이 조인 연산을 위해 윈도우 내의 각 스트림은 항상 메모리에 다 올라올 수 있다고 생각하고, 조인 과정에서 생산된 중간 결과와 최종 결과를 메모리에 유지하기 위해 차지하는 총 크기(total size)를 질의 계획의 실행 비용으로 간주한다. 이는 메모리에 올라온 데이타를 읽고 정렬하여 조인하는 시간보다 처리 과정에서 생산된 중간 결과와 최종 결과를 메모리에 쓰는 데 걸리는 시간이 월등히 크기 때문이다. 참고로, 본 논문에서 고려하는 비용 모델은 관계형 데이타베이스에서 조인 연산의 결과 크기를 추정하는 기법 [11]을 따른다. 연속 질의 i (CQ_i)의 실행 비용을 다음과 같이 나타낼 수 있다. $e\_cost(ep) = \mathbf{NI}(ep) + \mathbf{NF}(ep)$ [ep : CQ_i의 실행 플랜(execution plan) se : ep의 조인 연산에 대한 부분 표현 **NI** : ep의 실행으로 생성된 모든 중간 결과의 크기(추정 튜플 개수) **NF** : ep의 실행으로 생성된 최종 결과의 크기(추정 튜플 개수)] 본 논문의 모델을 바탕으로 다음의 CQ_i 와 CQ_j가 동시에 실행될 때, 각 연속 질의의 실행 비용은 다음과 같이 계산된다. A, B, C는 데이타 스트림 소스이고, CQ_i 의 실행 계획은 $A \bowtie B$, CQ_j 의 실행 계획은 $(A \bowtie B) \bowtie C$) 이라고 하자. CQ_i 의 실행 플랜에서 중간 결과 자체가 최종 결과가 되기 때문에, $NI(A \bowtie B) = 0$, $NF(A \bowtie B) = S(A) \times S(B) / \max(d(A), d(B))$가 된다. 여기서, $S(A)$ 는 질의가 정의한 윈도우 내에 존재하는 스트림 A의 평균 튜플 개수($S(B)$에 대해서도 동일하다), $d(A)$ 는 질의가 정의한 윈도우 내에 존재하는 스트림 A의 평균 튜플 개수 중에서 질의에서 관심 있는 애트리뷰트에 대해 서로 다른 값을 갖는 튜플의 개수이다($d(B)$에 대해서도 동일). $S(A) = 500$, $S(B) = 2{,}000$, $d(A) = 25$, $d(B) = 80$ 이라고 가정하면, CQ_i 의 실행 비용은 $e\_cost(A \bowtie B) = NI(A \bowtie B) + NF(A \bowtie B) = 0 + 500 \times 2{,}000 / \max(25, 80) = 1{,}000{,}000 / 80 = 12{,}500$ 이 되고, 동일하게 CQ_j 의 실행 비용은 $e\_cost(A \bowtie B \bowtie C) = NI(A \bowtie B) + NF((A \bowtie B) \bowtie C) = 12{,}500 + 12{,}500 \times 800 / \max(80, 64) = 150{,}000$ 이 된다.
중간 결과	알고리즘의 복잡성을 감안하여, 중간 결과들은 모두 이진 조인(binary join)으로 나타낼 수 있다고 본다. 중간 결과가 k(≥3)차 조인이라 하더라도 본 논문의 아이디어에 영향을 주지는 않는다.

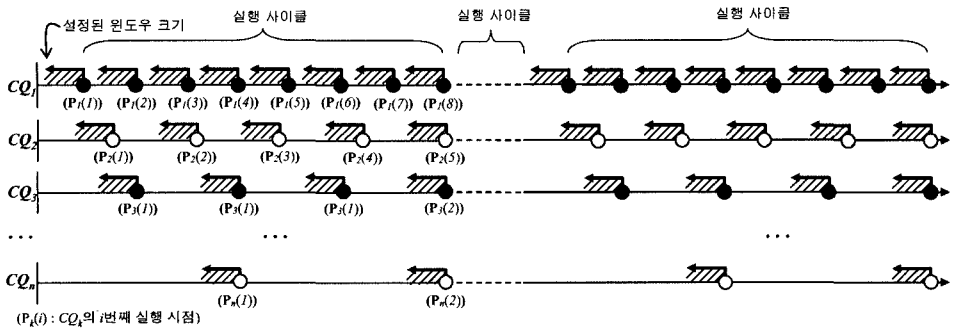

그림 4 n 개의 연속 질의의 실행

될 수 있다는 것이다. 그림 4의 CQ_1과 CQ_2의 실행을 보다 상세히 나타낸 그림 5를 살펴보자.

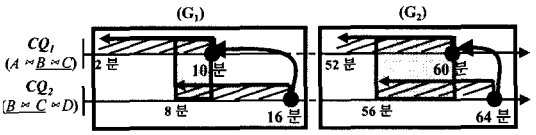

그림 5 중첩된 윈도우의 크기가 서로 다를 경우

만약 CQ_1이 $((A \bowtie B) \bowtie C)$, CQ_2가 $(B \bowtie (C \bowtie D))$를 각각 자체 최적 계획으로 갖는다면, 이들은 각각 자체 최적 계획만을 실행할 수도 있지만 중간 결과가 될 수 있는 $B \bowtie C$를 서로 공유하는 계획을 전체적으로 세울 수 있다. 데이타 스트림에서는 해당 윈도우가 완전히 겹치지 않는 한, 전체 중간 결과를 재사용 할 수 없고 일부의 중간 결과만을 재사용할 수 있다. 즉 그림 5-(G_1)에서는 중첩된 윈도우의 크기가 2분이고, 그림 5-(G_2)에서는 4분이기 때문에 CQ_1과 CQ_2는 $B \bowtie C$의 일부만을 공유할 수 있다. 중첩된 윈도우의 크기에 따라, (G_1)과 (G_2)의 전체 계획도 서로 달라질 수 있다. (G_1)에서는 2분 밖에 겹치지 않기 때문에 공유할 수 있는 $B \bowtie C$의 부분이 적으므로, $B \bowtie C$에 대해서 절약할 수 있는 실행 비용이 적다. 이때 $B \bowtie C$를 공유하는 계획보다 그림 6(a)와 같이 오히려 자체의 최적 계획을 개별적으로 실행하는 것이 효율적일 수가 있다. 한편, (G_2)에서는 4분이 겹쳐서 그림 6(b)와 같이 $B \bowtie C$를 공유하는 계획을 실행하는 것이 더 좋을 수 있다. 본 논문의 목적은 전체의 중간 결과는 물론, 기존 연구와는 달리 일부의 중간 결과까지도 재사용하는 것을 고려하여 최적에 가까운 전체 계획을 세우는 것이다.

그림 6의 (a), (b)는 스트림의 시간적인 요소를 고려하지 않은 기존 데이타베이스 기반의 개괄적 실행 트리이므로, 데이타 스트림 환경에서 정확히 어떤 부분을 재사용하는지를 나타내는 실행 트리의 구성이 필요하다. 그림 7은 이러한 목적에 맞추어 그림 5-(G_2)를 다시 구성한 트리이다. 연속 질의 CQ_1과 CQ_2는 $B_{[56,\ 60]} \bowtie C_{[56,\ 60]}$의 결과를 공유하는 계획으로 실행된다.

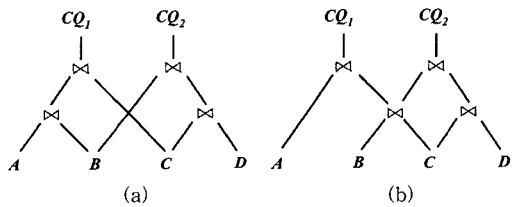

그림 6 그림 5의 (G_1)과 (G_2)의 개괄적인 전체 계획

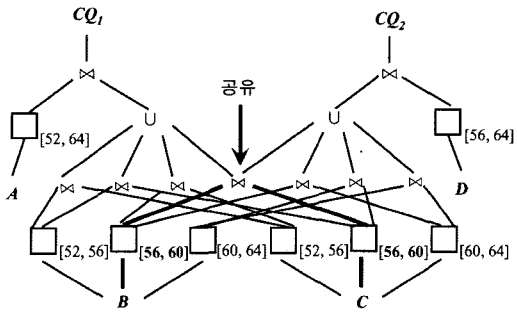

그림 7 그림 5의 (G_2)에서 실행될 구체화된 전체 실행 계획

마지막으로, 위에서 언급한 고려 사항들을 어느 시점까지 적용하느냐 하는 것에 대한 분석이 필요하다. 그림 4에서 보는 바와 같이 n 개의 질의가 동시에 실행되는 특정 시점 $(P_1(8) = P_2(5) = \cdots = P_n(2))$까지만 위의 사항들을 고려하여 각 실행 시점에서 질의들의 전체 계획을 세우면, 다음의 특정 시점이 될 때까지 세워진 그 전체 계획을 그대로 반복 실행할 수 있다. 이러한 특정 시점 사이의 구간을 본 논문에서는 시스템의 실행 사이클로 정의한다.

한편, 기존 연구에서 다뤄왔던 다중 질의의 최적화 문제의 복잡도가 NP-Hard 임이 증명되었는데[9], 본 논문에서는 매 실행 시점마다 다중 질의를 최적화하는 문제로 고려될 수 있으므로 기존 문제보다 더 높은 복잡도를 가진다. 그러므로, 탐욕 기반의 경험적 방법을 통해 문제를 해결하고자 한다.

4. 접근 방법

4.1 질의 실행 간격을 이용한 실행 사이클 찾기

본 논문에서는 모든 연속 질의들의 실행 간격의 최소 공배수로 실행 사이클을 설정한다. 즉 세 개의 연속 질의의 실행 간격이 각각 10분, 20분, 30분이라면 이 질의를 시스템이 처리하는 최초의 실행 시점인 '60분'이 시스템의 실행 사이클이 될 것이다.

4.2 질의 실행 시점들의 분류

보조 정리 1. n 개의 연속 질의가 실행될 때, 한 실행 사이클 내에서 전체적으로 최적인 실행 계획을 세우기 위해서 고려해야 할 실행 계획들의 개수는 다음과 같다.

$$\prod_{k=1}^{n} M_k^{(P_k)}$$

(M_k = CQ_k가 선택할 수 있는 실행 플랜의 수
P_k = 한 실행 사이클 내에서 CQ_k의 총 실행 시점의 개수)

증명. 한 실행 사이클 내에 존재하는 모든 CQ 들은

매 실행마다 다른 CQ 들이 생산한 중간 결과를 재사용할 수 있는지 여부를 판단해서 실행 계획을 세워야 한다. 따라서, 동일한 CQ_k가 반복해서 실행되더라도 다른 $CQ_{k'}$에 의해 이전에 실행된 CQ_k의 실행 계획과 다음에 실행될 CQ_k의 실행 계획이 서로 다를 수 있다. 또한, 그 실행 계획들이 포함된 전체 실행 계획을 세워야 하므로, CQ_k가 실행되는 매 시점마다 M_k 개의 경우를 고려해야 한다(즉, M_k^{Pk} 개). 이것을 n 개의 연속 질의를 위한 전체 실행 계획을 세우기 위해 동시에 고려해 주어야 하므로, 총 후보 실행 계획들의 개수는 $M_1^{P1} \times M_2^{P2} \times \cdots \times M_n^{Pn} = \prod_{k=1}^{n} M_k^{(P_k)}$ 이 된다.

보조 정리 1에 의해, P_k와 n이 커질수록 실행 사이클 내의 최적 전체 실행 계획을 구하는 것은 실질적으로 어려운 상황이다. 최적 전체 실행 계획에 근접하는 실용적인 전체 계획을 얻기 위해, 다음의 사항을 유심히 고려할 필요가 있다.

그림 5의 (G_2)를 살펴보자. '60분'에 실행되는 CQ_1과 '64분'에 실행되는 CQ_2는 서로 56분~60분 사이의 윈도우가 겹치며, $B \bowtie C$라는 부분 조인 표현을 공통으로 갖는다. 각 질의의 지역 최적 계획으로 구성된 전체 계획보다 CQ_1과 CQ_2의 공유 계획으로 구성된 전체 계획이 더 효율적이라면 후자를 전체 계획으로 결정할 수 있다. 만약 CQ_1과 CQ_2의 윈도우가 서로 겹치지 않거나 공통의 부분 조인 표현이 존재하지 않는다면, 그들 간의 공유 계획은 존재하지 않으므로, CQ_1과 CQ_2의 전체 계획은 지역 질의-처리 계획만으로 구성될 것이다. 이처럼, 윈도우가 중첩되지 않거나 공통의 부분 조인 표현을 갖지 않는 시점에서 실행될 질의에 대해서는 공유 계획을 고려하지 않고 전체 계획을 세우는 것이 바람직하다. 예를 들어, 그림 5의 (G_2)의 '60분'과 '64분'은 서로 '관련 있는 시점'이다. 여기서, '관련 있는 시점'이라는 의미는 그 시점에서 실행될 질의들은 윈도우가 겹쳐 있고 공통의 부분 조인 표현을 가지고 있어서, 서로 간에 공유 계획을 세울 것을 고려할 필요가 있다는 것이다. 본 논문에서는 한 실행 사이클 내에 있는 모든 시점들을 '관련 있는 시점' 들만의 집합으로 묶고, 집합 내의 시점에서 실행될 질의들에 대해서만 전체 계획을 고려한 다음, 각 집합 내에서 얻어진 계획을 병합함으로써 실행 사이클 내의 전체 계획을 얻는다.

정의 3. 한 실행 사이클 내에 있는 질의 실행 시점들이 "관련 있는 시점" 들로 분류된 집합을 ***SRP***(maximal Set of Related execution Points)라고 정의한다.

그림 8은 그림 4의 $CQ_1 \sim CQ_3$의 실행만을 고려했을 때, 찾아낸 SRP 들을 보여준다. 하나의 SRP 안에 있는 "관련 있는 시점"들끼리 서로 선으로 연결된 것을 볼 수 있다.

4.3 재사용 비율과 이익

하나의 SRP에 있는 다수의 시점에서 실행될 연속 질의의 전체 계획을 고려하는 것도 쉽지 않다. 실행 사이클 내의 모든 시점들이 하나의 SRP에 속해 있다면, 보조 정리 1에 의해 최적 전체 계획을 결정하는 것은 거의 불가능해 진다. 이 절에서는 본 논문에서 제안하는 알고리즘의 기저가 될 재사용 비율과 이익을 정의한다.

정의 4. SRP 내의 각 시점에서 실행될 연속 질의들이 공유 가능한 전체 또는 일부의 중간 결과를 재사용함으로써 그 결과에 대한 실행 비용을 얼마나 절약할 수 있는지를 나타낸 비율을 **재사용 비율(Reusable Ratio)**이라고 정의한다.

재사용 비율은 다음과 같이 표현된다.

$$\text{SRP 내의 } P_A(i) \text{에서 실행될 } CQ_A \text{에 대해 } P_B(j) \text{에서 실행될 } CQ_B \text{가 공유 가능한 중간 결과(k-차 조인)를 재사용할 수 있는 비율 } (P_A(i) < P_B(j)) = \left[\frac{\{P_A(i) - (P_B(j) - CQ_B \text{의 윈도우 크기})\}}{(CQ_A \text{의 윈도우 크기})}\right]^k \times 100$$

그림 9와 같이 동일한 SRP 내에 존재하는 P_A에서 실행될 CQ_1과 P_B에서 실행될 CQ_2가 주어져 있다고 하자.

CQ_2가 CQ_1과 공통으로 참조하는 스트림 데이타에 대한 중첩 비율은 다음과 같다.

$$CQ_2\text{가 } CQ_1\text{과 공통으로 참조하는 스트림 데이터에 대한 중첩 비율} = \frac{12 - (18 - 10)}{10} \times 100 = 40\%$$

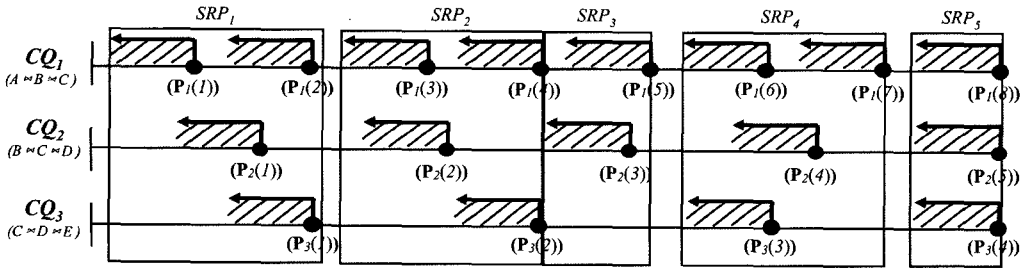

그림 8 실행 사이클 내에 있는 SRP로 분류된 실행 시점들

CQ_1	(1, A ⋈B, 10 min, 12 min)
CQ_2	(2, A ⋈B ⋈C, 10 min, 18 min)

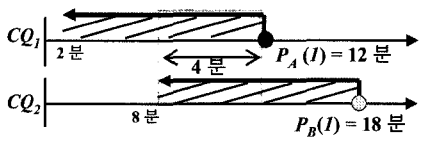

그림 9 동일한 SRP 내에 존재하는 P_A 와 P_B

이 비율의 의미는 CQ_2의 윈도우 내에 존재하는 스트림 A와 스트림 B의 데이타는 CQ_1의 윈도우 내에 존재하는 스트림 A와 스트림 B의 데이타 중에 각각 40%가 중첩되어 있다는 것이다. 만약 CQ_1과 CQ_2가 중간 결과인 $A \bowtie B$를 공유하는 전체 계획을 세웠다고 하자. 그렇다면, CQ_2는 CQ_1이 생산한 전체 중간 결과, $A \bowtie B$ 중에서 $A_{[8,\ 12]} \bowtie B_{[8,\ 12]}$를 재사용함으로써, 그것에 대한 실행비용을 절약할 수 있다. 그러므로 CQ_1에 대한 CQ_2의 재사용 비율은 스트림 A의 중첩 비율(40%) × 스트림 B의 중첩 비율(40%) = 16%가 된다.

SRP에 있는 각 시점에서 실행될 연속 질의에 대해 설정된 재사용 비율은 지역 질의-처리 계획에 비해 조정 질의-처리 계획을 선택했을 때의 이익(benefit)을 계산하기 위해 활용될 것이다. 여기서 이익이란 지역 질의-처리 계획에 비해 재사용이 가능한 중간 결과를 공유하는 조정 질의-처리 계획을 선택했을 때 절약할 수 있는 실행 비용이다. 이익을 계산하는 공식은 5.2.4 절에서 제시될 것이다.

끝으로 이익이 가장 큰 중간 결과를 갖는 공유 계획을 SRP 안에서 구성함으로써 실행 사이클 내의 전체 계획을 세움으로써 본 논문의 작업을 완성한다.

5. GAGPC(A Greedy Algorithm for a Global Plan Construction)

본 논문의 알고리즘인 GAGPC를 제시한다. 그것은 크게 5.1과 5.2로 나뉘어진다.

5.1 실행 사이클 결정

시스템에 전달된 n 개의 연속 질의의 실행 간격을 분석하여 반복된 실행 사이클을 찾아내는 단계이다.

$EC = LCD(CQ_1.T,\ CQ_2.T,\ CQ_3.T,\ \cdots,\ CQ_n.T)$

[EC: 실행 사이클, LCD는 연속 질의들의 실행 간격에 대한 최소 공배수를 계산하는 함수]

5.2 모든 질의에 대한 전체 계획 구성

실행 사이클(EC) 내에 있는 각 시점 $P_k(i)$에서 실행될 연속 질의의 계획을 구성함으로써, 전체 계획을 결정한다(이 절에서 $P_k(i)$를 각 CQ_k에 대한 구분 없이 P_n으로 표기한다).

5.2.1 사이클 내의 모든 실행 시점을 위한 정보 생성

각 실행 시점(P_n)을 위한 데이타 구조는 다음과 같다.

```
struct Pn {  // Pk(i) == Pn
  int k;     // 질의의 식별자(CQ identifier)
  int W;     // 윈도우 크기(window size)
  int T;     // 실제 실행 시간(actual execution time)
  double** RR; // 다른 질의와의 재사용 비율(reusable ratio)
};
```

그림 10은 정의된 데이타 구조를 바탕으로 모든 실행 시점의 정보를 구성하는 프로시저이다. 이 프로시저의 시간 복잡도는 $O(n \times E)$이다. 이때 n은 시스템에 전달된 연속 질의의 개수(**CQ** 집합의 원소 개수)이고, E는 한 사이클 내에서 실행 간격이 가장 짧은 질의의 실행 횟수이다.

5.2.2 실행 사이클 내에 존재하는 SRP 찾기

다음과 같이 주어진 CQ_a, CQ_b, CQ_c가 있다고 가정해 보자.

CQ_a	(a, A ⋈B ⋈C, 5 분, 10 분)	CQ_b	(b, A ⋈B, 10 분, 18 분)	CQ_c	(c, B ⋈C, 8 분, 30 분)

```
Procedure CONSTR_PNT
Input : CQSet -Set of continuous queries delivered to a system
        EC - Execution cycle among CQ
Output : p_list - Linked_list to store execution points
BEGIN
1     WHILE( CQSet ≠ Φ ) {
2       Get a continuous query, CQ, from CQSet
3       Calculate the total number of execution points of CQ in EC
4       Find data of all execution points of CQ of through 'FOR' loop
5       Add them to p_list in order
6       CQSet = CQSet – { CQ }
7     }
8     RETURN(p_list)
END
```

그림 10 프로시저 CONSTR_PNT

그림 11은 주어진 연속 질의들에 대한 SRP들을 찾아낸 것이다. 그림 11에서 선으로 연결된 시점들만 서로 "관련되어" 있고, 다른 시점으로의 연결이 끊어진 시점까지 하나의 SRP를 구성한다. 즉, $P_a(2)$와 $P_a(3)$ 사이의 연결이 끊어져 있는데, $P_a(2)$까지의 시점들이 SRP_1을 구성하고, $P_a(3)$ 이후의 시점들이 SRP_2를 구성하게 된다.

그림 12는 각 SRP를 찾아내는 프로시저이다. 그림 12의 프로시저를 분석해 보면, p_list의 P_n들을 정렬하고(Line 1), p_list의 원소를 한 번씩 다 읽고(Line 4), 발견된 SRP가 서로 결합될 수 있는지를 확인하므로

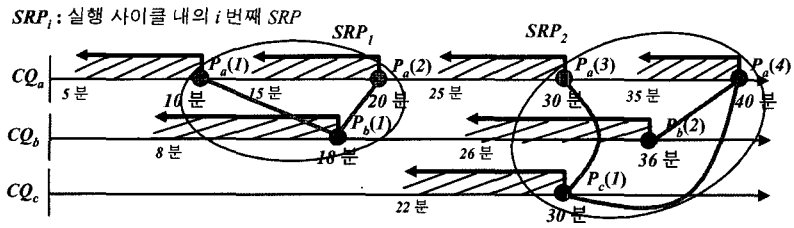

그림 11 실행 사이클 내에서 발견된 SRP_1과 SRP_2

```
Procedure FIND_SRP
Input : p_list - Linked_list to store execution points
Output : SRP_LIST - Linked_list to store each SRP
BEGIN
1     Sort all execution points of p_list in non-decreasing order on their actual execution times
2     Get the first element from sorted p_list
3     Append it to a SRP
4     WHILE (the current element is not the last element of p_list) {
5       Get the next element from p_list
6       IF ({join expressions of CQs executed at the current element and the next element are common
7             && their window is overlapped each other} )
8         Append the next element to a SRP because they are in the same SRP
9       ELSE
10        Append a obtained SRP to SRP_LIST
11      Set the next element to the current element
12    }
13    Check if all SRPs in SRP_List are merged between each other
14    RETURN(SRP_LIST)
END
```

그림 12 프로시저 FIND_SRP

(Line 14) 시간 복잡도는 O(NlogN + N + N)) = O(N(logN + 2))가 된다. 이 때, N은 *p_list* 의 원소 개수(실행 사이클 내에 있는 모든 P_n의 개수)이다.

5.2.3 SRP에서 질의 간 윈도우 중첩 비율 계산

그림 13은 P_n에서 0으로 초기화된 질의 간 윈도우 중첩 비율을 구하는 프로시저이다. 이 프로시저를 분석해 보면, 시간 복잡도는 O(K(2plogp + (p^2 - p)/2))이 된다. 이때, K는 실행 사이클 내의 *SRP* 의 개수, p는 *SRP* 내의 최대 실행 시점의 개수이다.

5.2.4 각 SRP에 있는 각 시점에서 실행될 연속 질의의 실행 계획 작성

*GAGPC*는 마지막으로 모든 *SRP*에 대해서 다음의 단계 1~단계 3을 통해 하위-전체 계획들(sub-global plans)을 세우고 그것들을 합병함으로써, 하나의 전체

```
Procedure CALC_RR
Input : SRP_LIST, List to store SRPs
BEGIN
1     FOR EACH SRP in SRP_LIST
2       Sort all execution points in SRP by non-increasing order on their actual execution times
3       FOR EACH sorted execution point in order
4         WHILE (SRP ≠ Φ // the identifier of the current element is not equal to the identifier of the next element ) {
5           IF 'window beginning time' of CQ to be executed at the current element is earlier than that of the next element
6             /* We can reuse the whole intermediate results of CQ previously executed */
7             Calculate the reusable ratio of the current element on the next element using only window size of each other
8           ELSE
9             /* We can reuse the partial intermediate results of CQ previously executed */
10            Calculate the reusable ratio of the current element on the next element using window size and actual execution times of each other
11          Remove the next element in SRP
12          Get the next element ;
13        }
14      END FOR EACH
15      Sort all execution points in SRP by non-decreasing order on their actual execution times
16    END FOR EACH
END
```

그림 13 프로시저 CALC_RR

계획을 획득한다.

정의 5. 연속 질의의 조인에 대한 부분 표현(sub-expression)을 조인 부분(join fragment)이라고 정의한다.

예를 들어 $A \bowtie B \bowtie C$의 경우 조인 부분들은 $A \bowtie B$, $B \bowtie C$ 등이 된다.

단계 1은 *SRP* 안에 있는 각 시점(P_s)에서 실행될 모든 질의에 대해, 공통인 조인 부분의 집합(GFR)을 찾아내는 과정이다. 그림 14는 단계 1의 프로시저이고, 그것의 시간 복잡도는 $O(p \times p) = O(p^2)$가 된다. 이때, p는 *SRP* 안에 있는 P_s의 개수이다.

단계 2. 단계 2는 *GFR* 중에서 *SRP* 내에서 다중 연속 질의들이 공유할 조인 부분들을 선택하는 과정이다. 이때, *GAGPC*는 각 루프에서 가장 높은 이익을 갖는 조인 부분(FR_m)을 고른다. 각 루프에서 FR_m이 선택된 후, *GFR*에 남아 있는 각 원소에 대한 이익을 재계산해야 하는데, 왜냐하면 FR_m가 대립되는(conflict) 원소의 이익이 줄어들 수 있기 때문이다. 예를 들어 CQ_i의 조인 표현이 $A \bowtie B \bowtie C$와 같이 주어졌다고 하자. 그것의 조인 부분은 $A \bowtie B$와 $B \bowtie C$가 된다. 위의 FR_m이 $A \bowtie B$라고 한다면, 대립되는 조인 부분인 $B \bowtie C$는 CQ_i의 실행 시 생성되는 중간 결과가 될 수 없으므로, $B \bowtie C$에 대한 이익은 줄어 들게 되는 것이다. 만약 FR_m의 이익이 0보다 같거나 작다면, *GFR*에 있는 모든 원소들을 재사용하더라도 더 이상 실행 비용을 줄일 수 없다는 의미이므로, *GAGPC*는 단계 3으로 진행한다.

다음은 각 조인 부분에 대한 이익을 계산하는 공식이다.

$$benefit(FR_k) = \sum_{CQ_m \in SCQ} [\, e\_cost(LQP(CQ_m)) - \min\{\, e\_cost(AQP(CQ_m)) + e\_cost(FR_k \text{ of } CQ_m) \times \mathbf{Reusable\_Ratio}(CQ_m, CQ_n)\}](CQ_i \neq CQ_j)$$

$\{FR_k : SRP$에 있는 시점에서 실행될 질의의 각 조인 부분
$SCQ : FR_k$를 가지고 있는 CQ_i들의 집합
$CQ_m : SRP$에 있는 시점에서 실행되는 m번째 CQ_i, $CQ_n : SRP$에 있는 시점에서 실행되는 n번째 CQ_j
$LQP(CQ_i) : CQ_i$의 지역 질의-처리 플랜
$AQP(CQ_i)) : FR_k$를 공유하는 CQ_i의 조정 질의-처리 플랜
$e\_cost(query\ plan)$: $query\ plan$의 실행 비용
$\mathbf{Reusable_Ratio}(CQ_m, CQ_n) : CQ_n$에 대한 CQ_m의 재사용 비율(reusable ratio)]

그림 15는 단계 2의 프로시저이고, 그것의 시간 복잡도는 $O(f \times (f \times p \times (p-1))) = O(f^2 \times p^2)$이 된다. 이때, f는 *GFR*의 원소 개수, p는 *SRP* 안에 있는 시점의 개수이다.

단계 3. 단계 2가 종료되면 P_s에서 실행될 질의가 재사용할 조인 부분들이 모두 결정된다. 각 시점에서의 실행 계획에 이 조인 부분을 포함시킨다. 어떤 질의가 선택된 조인 부분들을 포함하지 않는다면, 그것의 지역 질의-처리 계획이 실행될 것이다. 그림 16은 단계 3의 프로시저이다. 그것의 시간 복잡도는 $O(p)$가 된다. p는 *SRP* 안에 있는 시점의 개수이다.

6. 동작 예제 및 시뮬레이션

이 장에서는 다중 연속 질의에 대한 전체 계획을 얻는 과정을 동작 예제를 통해 살펴보고, 실제 시뮬레이션 결과를 통해 제안된 방법의 우수한 성능을 제시한다.

6.1 동작 예제

다음의 연속 질의들이 주어졌다고 하자.

CQ_1	(1, A =B =C, 8 min, 12 min)	CQ_2	(2, B =C =D, 10 min, 15 min)	CQ_3	(3, C =D =E, 4 min, 30 min)
CQ_4	(4, A =B, 10 min, 20 min)	CQ_5	(5, D =E, 10 min, 30 min)	CQ_6	(6, B =C, 10 min, 60 min)

위의 연속 질의들의 실행은 그림 17과 같이 기술된다. 그림에서 왼쪽 부분은 각 CQ_k의 조인 그래프이다. 예를 들어 CQ_1의 조인 그래프는 스트림 A와 B, 스트림 B와 C가 조인되지만, 스트림 A와 C는 조인되지 않는다는 것을 의미한다.

먼저, SRP_1에서 실행되는 질의들의 전체 계획을 결정해 보자.

단계 1에서, $GFR = \{A \bowtie B, B \bowtie C, C \bowtie D, D \bowtie E\}$을 얻을 수 있다.

단계 2에서, *GFR*의 각 원소에 대한 이익을 구한다. $FR_1 = A \bowtie B$, $FR_2 = B \bowtie C$, $FR_3 = C \bowtie D$, $FR_4 = D \bowtie E$ 등이 *GFR*의 원소가 된다. FR_1이 *GFR*에서 가장 높은 이익을 갖는 조인 부분이라고 하

```
Procedure STEP1
Input : SRP, An item of SRP_LIST
Output : GFR(Global FRagments) , A set of all common join-fragments of all CQ in SRP
//  LFR(Local FRagments) :  Set of all join-fragments of a CQ in SRP
BEGIN
1       GFR = Φ
2       FOR EACH a CQ to be executed at each Ps in SRP
3         We get LFR1 = { all join-fragments of CQ };
4         FOR EACH CQ' to be executed at each Ps' in SRP (Ps ≠ Ps')
5           We get LFR2 = { all join-fragments of CQ' };
6           GFR = GFR ∪ (LFR1 ∩ LFR2);
7         END FOR EACH
8       END FOR EACH
9   RETURN(GFR);
END
```

그림 14 프로시저 STEP1

```
Procedure STEP2
Input : SRP, GFR
Output : SFR(Selected FRagments) - List of the selected join-fragments to reuse in SRP
/* Reusable_Ratio(CQ , CQ'); Reusable Ratio of CQ on CQ' */
BEGIN
1     SFR = Φ;
2     WHILE (GFR ≠ Φ) {
3       FOR EACH FR_k in GFR
4         FOR EACH CQ to be executed at an execution point in SRP
5           FOR EACH CQ' to be executed at an execution point in SRP
6             IF( CQ and CQ' has FR_k && their identifiers are not the same)
7               Calculate the benefit when CQ reuses FR_k using Reusable_Ratio(CQ , CQ');
8           END FOR EACH
9         END FOR EACH
10      END FOR EACH
11      Get the join-fragment, FR_m , with the highest benefit among join-fragments of GFR;
12      IF (the benefit of FR_m is less than or equal to 0 ) break;
13      ELSE{
14        SFR appends FR_m; GFR = GFR - {FR_m};
15      }
16    }
17    RETURN(SFR);
END
```

그림 15 프로시저 STEP2

```
Procedure STEP3
Input : SFR, Set of selected join-fragments to reuse in SRP
/*  LQP(CQ) : Local Query-processing Plan of CQ
    FQP(CQ) : Final Query-processing Plan of CQ  */
BEGIN
1     FOR EACH CQ to be executed at an execution point in SRP
2       We get LFR = { all join-fragments of CQ };
3       IF ( LFR ∩ SFR ≠ Φ ) {
4         Construct FQP(CQ) with join-fragments in SFR
5       } ELSE {
6         FQP(CQ) = LQP(CQ)
7       }
8     END FOR EACH
END
```

그림 16 프로시저 STEP3

자. 본 논문의 알고리즘에 의해, GFR에서 FR_1이 제거되고 나머지 원소에 대한 이익이 재계산된다. 다음, FR_3이 가장 높은 이익을 갖는 원소라면 GFR에서 FR_3이 제거되고, 남아 있는 원소에 대한 이익이 재계산된다. 마지막으로, FR_2가 가장 높은 이익을 갖는다면 GFR에서 FR_2가 제거되고, 남아 있는 FR_4의 이익이 재계산된다. 만약, 그 이익이 0이 되면 GFR에서 더 이상 고려할 수 있는 원소가 없으므로, 단계 3을 진행한다.

단계 3에서, 연속 질의들의 전체 계획을 얻는다. SRP_1에서는 가장 높은 이익을 갖는 순서대로 $A \bowtie B$, $C \bowtie D$, $B \bowtie C$가 공유될 조인 부분으로 선택되었다. 먼저 $A \bowtie B$에 대해, (P_1), (P_3), (P_4), (P_8), (P_9)에서 실행될 질의들은 $A \bowtie B$의 일부를 공유하는 조정 질의-처리 계획을 실행 계획으로 결정한다. 그 다음, $C \bowtie D$에 대해, (P_5), (P_6)에서 실행될 질의들은 $C \bowtie D$의 일부를 공유하는 조정 질의-처리 계획을 실행 계획으로 결정한다. 다음, $B \bowtie C$에 대해, (P_{10}), (P_{11})에서 실행될 질의들은 $B \bowtie C$의 일부를 공유하는 조정 질의-처리 계획을 실행 계획으로 결정한다. (P_1)에서 $A \bowtie B$

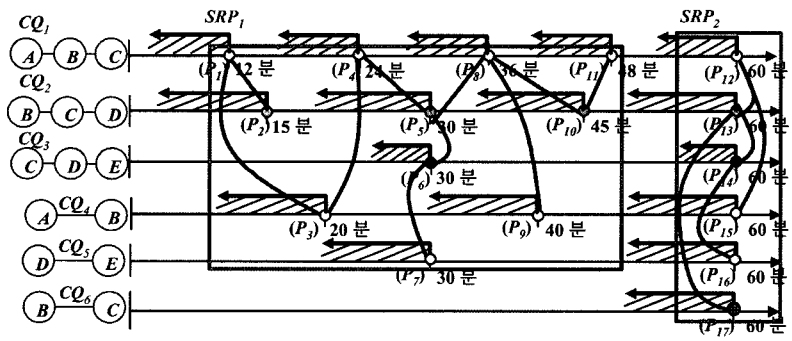

그림 17 $CQ_1 \sim CQ_6$의 실행: 각 질의의 실행 시점들은 SRP_1과 SRP_2로 나뉘어 진다.

의 일부가 실행 계획에 이미 포함되었으므로, (P_2)에서는 그것의 지역 질의-처리 계획이 실행 계획으로 선택된다. 그리고 단계 2에서 얻은 조인 부분들을 전혀 갖지 않는 (P_7)에서는 그것의 지역 질의-처리 계획이 실행 계획으로 결정될 것이다. 동일하게, SRP_2에서 실행될 전체 계획을 구성할 수 있다.

그림 18은 SRP_1의 전체 계획의 일부를 나타낸 것이다. 그림 18(a)는 (P_5)와 (P_6), 그림 18(b)는 (P_8)과 (P_9)에서 실행될 전체 계획이다.

6.2 시뮬레이션

6.2.1과 6.2.2에서는 10개의 동일한 연속 질의 집합(QSET)에 대하여, (ㄱ) 본 논문의 GAGPC로 얻는 방법, (ㄴ) 모든 질의의 지역 질의-처리 계획만으로 구성(No Reusing Work)하여 얻는 방법, (ㄷ) 최적 전체 계획을 얻기 위해, 시간 제한을 두지 않고 철저한 탐색(exhaustive search)을 수행하여 얻는 방법, (ㄹ) NiagaraCQ or CACQ 등의 기존 연구의 알고리즘을 본 논문에서 다루고 있는 환경에 적용하여 질의 간에 윈도우가 완전히 겹치는 경우에만 중간 결과의 재사용이 가능한 것으로 하여 얻는 방법 등으로 전체 계획을 구하였다. 그리고 윈도우 크기, 실행 간격, 연속 질의의 수 등을 매개변수로 하여 위의 각 방법으로 얻은 전체 계획의 실행 비용의 변화 추세를 그래프로 나타내었다. 한편, 6.2.3와 6.2.4에서는 질의의 수를 변화시켜, 각 전체 계획의 생성 시간 및 실행 비용의 변화 추세를 그래프로 나타내었다.

6.2.1 윈도우 크기에 따른 각 전체 계획의 실행 비용 변화 및 결과 분석

표 2는 실험에 사용된 각 질의 집합에 대한 상세한 설명이다. 집합 내 질의들의 윈도우 크기는 집합에 설정된 실행 간격 이하의 난수(random value)로 질의마다 다르게 설계하였다.

그림 19~21은 (ㄱ)의 실행 비용을 (ㄴ)~(ㄹ)과 비교한 그래프이다. 그림 19에서는 (ㄱ)이 전체적으로 6~8% 정도 (ㄴ)의 실행 비용을 줄이는 경향을 보였다. 그림 20에서는 (ㄱ)이 최적 계획을 얻는 (ㄷ)과 동일한 결과를 보여 최적 계획을 얻는데 상당히 유용함을 알 수 있다. 그림 21에서는 (ㄱ)이 (ㄹ)에 비해 2~5% 정도의 성능 향상을 보여 GAGPC가 기존 연구의 알고리즘보다 더 우수하다고 할 수 있겠다.

6.2.2 실행 간격에 따른 각 전체 계획의 실행 비용 변화 및 결과 분석

표 3은 실험에 사용된 각 질의 집합에 대한 상세한 설명이다. 집합 내 질의들의 실행 간격은 집합에 설정된 윈도우 크기보다 큰 난수(random value)로 질의마다 다르게 설계하였다.

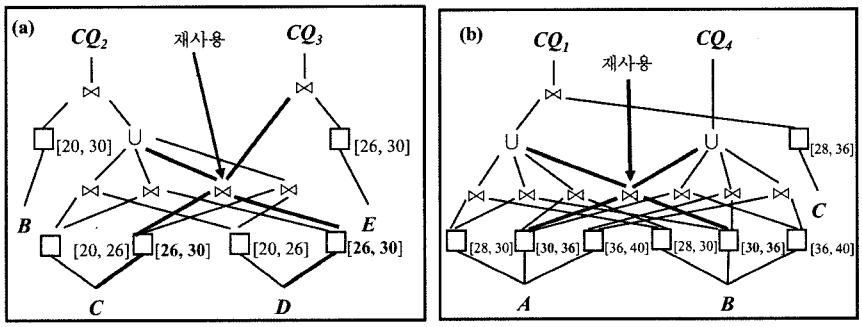

그림 18 (P_5)와 (P_6), (P_8)과 (P_9)에서 실행될 전체 계획

표 2 실험에 사용된 각 질의 집합에 대한 상세 설명 I

질의 집합	질의의 수	조인 조건 :동치-조인(equi-join)	윈도우 크기	실행 간격
QSET1	5	$A \bowtie B \bowtie C$	해당 실행 간격 이하의 난수	600
QSET2	5	$A \bowtie B \bowtie C$	해당 실행 간격 이하의 난수	700
...				
QSET10	5	$A \bowtie B \bowtie C$	해당 실행 간격 이하의 난수	1500

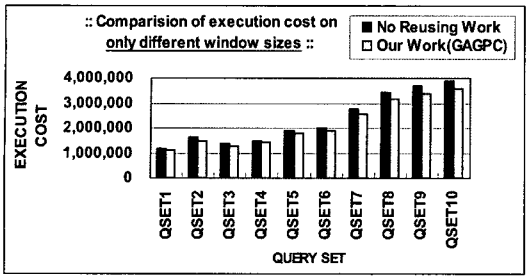

그림 19 (ㄱ)과 (ㄴ)의 실행 비용 비교

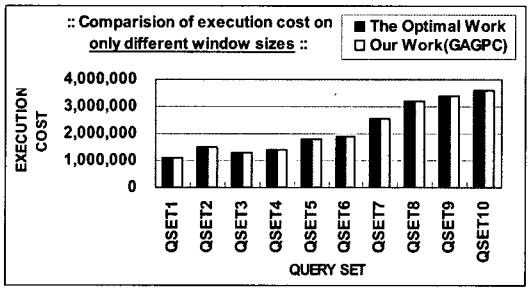

그림 20 (ㄱ)과 (ㄷ)의 실행 비용 비교

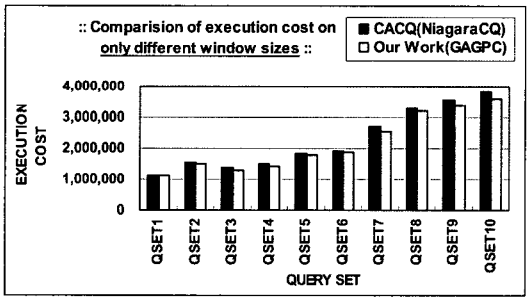

그림 21 (ㄱ)과 (ㄹ)의 실행 비용 비교

그림 22~24는 GAGPC를 통해 얻은 계획(ㄱ)의 실행 비용을 (ㄴ)~(ㄹ)과 비교한 그래프이다. 그림 22에서는 (ㄱ)이 전체적으로 4~6% 정도 (ㄴ)의 실행 비용을 줄

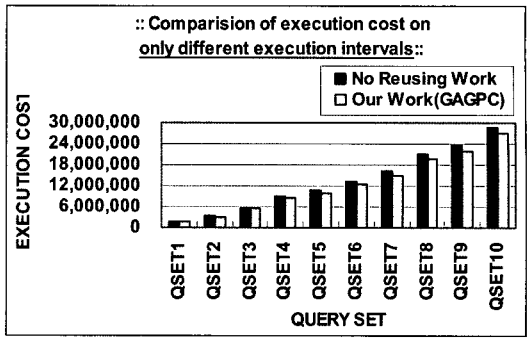

그림 22 (ㄱ)과 (ㄴ)의 실행 비용 비교

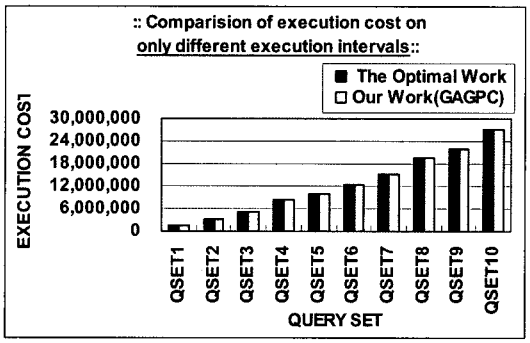

그림 23 (ㄱ)과 (ㄷ)의 실행 비용 비교

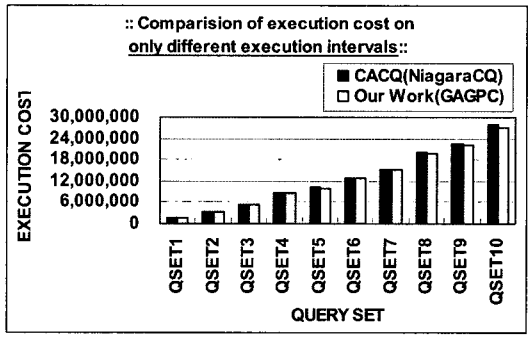

그림 24 (ㄱ)과 (ㄹ)의 실행 비용 비교

표 3 실험에 사용된 각 질의 집합에 대한 상세 설명 II

질의 집합	질의의 수	조인 조건 :동치-조인(equi-join)	윈도우 크기	실행 간격
QSET1	5	$A \bowtie B \bowtie C$	{10, 20, 30} 중의 하나	해당 윈도우 크기 이상의 난수
QSET2	5	$A \bowtie B \bowtie C$	{10, 20, 30} 중의 하나	해당 윈도우 크기 이상의 난수
...				
QSET10	5	$A \bowtie B \bowtie C$	{10, 20, 30} 중의 하나	해당 윈도우 크기 이상의 난수

이는 경향을 보였다. 그림 23에서는 (ㄱ)이 최적 계획을 얻는 (ㄷ)과 동일한 결과를 보여, GAGPC가 최적 계획을 얻는데 유용함을 앞선 결과에 이어 입증하였다. 그림 24에서는 (ㄱ)이 (ㄹ)에 비해 0.7~3.0% 정도의 성능 향상을 보였는데, 얻어진 전체 플랜의 실행 비용 자체가 워낙 크므로 비교적 GAGPC가 기존 연구의 알고리즘보다 더 우수하다고 할 수 있겠다.

6.2.3 질의 수의 변화에 따른 각 전체 계획의 생성 시간 및 결과 분석

표 4는 실험에 사용된 질의들에 대한 상세한 설명이다. 윈도우 크기, 실행 간격, 조인 조건을 동일하게 설정하고 질의 수를 증가시켜 시뮬레이션 하는 것은 무의미하므로, 질의가 추가될 때 마다 이들을 임의로 설정하여 질의 수 변화에 따른 각 전체 계획의 생성 시간을 측정하였다.

그림 25~26은 GAGPC를 통해 얻은 계획(ㄱ)의 생성 시간을 (ㄴ), (ㄹ)과 비교한 그래프이다. 보조 정리 1에 의해, 질의 개수가 증가할수록 최적 계획을 구하기 쉽지 않으므로 (ㄷ)과의 비교는 생략하였다. 그림 25, 26과 같이, (ㄱ), (ㄴ), (ㄹ) 모두 질의의 개수가 증가할수록 생성 시간은 다소 길어졌으나, (ㄱ), (ㄴ), (ㄹ) 사이의 시간 차이는 크게 나타나지 않았다. 이는 윈도우된 데이타 스트림이 모두 메모리 내에서 처리되기 때문에, 각 전체 계획을 생성하는 데 걸린 시간이 무시될 수 있을 만큼 짧았고 서로 엇비슷했던 것으로 풀이된다.

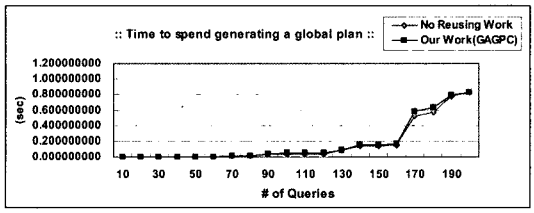

그림 25 (ㄱ)과 (ㄴ)의 전체 계획 생성 시간 비교

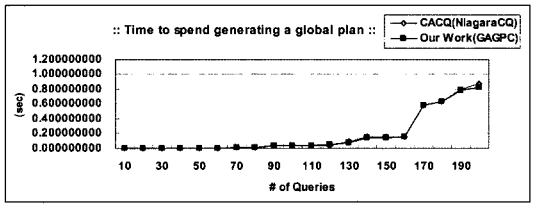

그림 26 (ㄱ)과 (ㄹ)의 전체 계획 생성 시간 비교

6.2.4 질의 수의 변화에 따른 각 전체 계획의 실행 비용 변화 및 결과 분석

표 4를 통해 얻은 질의들의 개수를 변화시키면서 각 전체 계획의 실행 비용을 측정하였다. 그림 27~28은 GAGPC를 통해 얻은 계획(ㄱ)의 실행 비용을 (ㄴ), (ㄹ)과 비교한 그래프이다. 6.2.3에서 언급한 바와 같이, (ㄷ)과의 비교는 생략하였다. 그림 27에서 질의의 개수가 증가할수록 (ㄱ)이 (ㄴ)의 실행 비용을 9~14% 가량 줄이는 경향을 보였다. 그림 28에서는 (ㄱ)이 (ㄹ)에 비해 4~6% 정도의 비교적 우수한 성능을 보였다. 따라서, 질의의 수가 증가하더라도 GAGPC의 확장성(scalability)이 보장되고 있음을 알 수 있다.

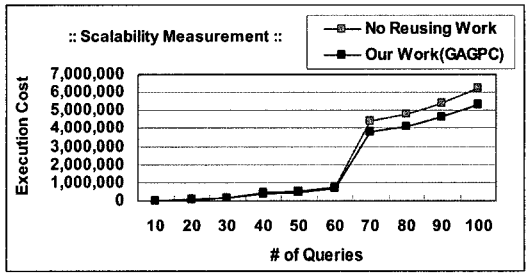

그림 27 (ㄱ)과 (ㄴ)의 실행 비용 비교

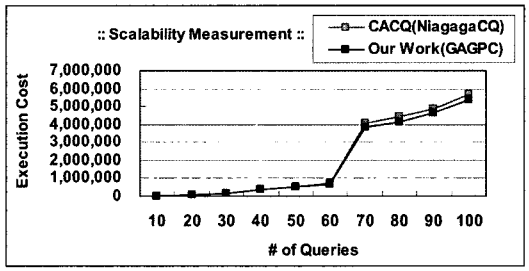

그림 28 (ㄱ)과 (ㄹ)의 실행 비용 비교

7. 결론 및 향후 연구

본 논문에서는 데이타 스트림이라는 환경에서 새로이 등장한 윈도우 크기와 실행 간격을 기반으로 다중 연속 질의의 전체 실행 계획을 구성하는 알고리즘을 제안하였다. 본 논문의 알고리즘에서는 크게 두 단계의 과정으로 다중 연속 질의들을 위한 전체 계획을 구성하였다. 먼저 다중 연속 질의의 실행 사이클을 알아내었다. 다

표 4 6.2.3의 실험에 사용된 질의의 상세 설명

질의	조인 조건 :동치-조인(equi-join)	윈도우 크기(min)	실행 간격(min)
CQ_i	$\{A \bowtie B \bowtie C, B \bowtie C \bowtie D, C \bowtie D \bowtie E\}$ 중의 하나	질의 개수 변화에 따라 취한 난수	해당 윈도우 크기 이상의 난수

음, 본 논문에서 제안된 재사용 비율과 이익 모델을 바탕으로 각 *SRP*에 있는 시점에서 실행될 연속 질의들의 전체 계획을 구성하였다. 이것은 세부적으로 단계 1~단계 3의 과정을 거쳐서 진행되었다. 이때, 가장 높은 이익을 갖는 조인 부분부터 포함시키는 탐욕 기반의 경험적 기법을 적용하여, 하나의 *SRP* 내에서 공유될 조인 부분을 선택하였다. 본 논문에 제시된 알고리즘은 매우 실용적이며, 최적 전체 계획에 거의 근접하는 전체 계획을 얻는 우수한 성능 결과를 시뮬레이션을 통해 알아보았다.

향후, 본 논문의 알고리즘은 조인 부분이 삼차(three-way) 이상일 때와 스트림 데이타의 변화율이 빈번하게 바뀌는 것 등을 고려하여 다양한 관점에서 분석될 필요가 있다.

참 고 문 헌

[1] Lukasz Golab M. Tamer Ozsu, Issues in data stream management, ACM SIGMOD Record, Volume 32, Issue 2, June 2003.

[2] S. Madden, and M.J. Franklin. Fjording the Stream: An Architecture for Queries over Streaming Sensor Data, Proc. ICDE, pp. 555-566, 2002.

[3] S. Madden, M. Shah, J.M. Hellerstein, and V. Raman. Continuously Adaptive Continuous Queries over Streams, Proc. ACM SIGMOD, pp. 49-60, 2002.

[4] Y.Yao and J.Gehrke. Query Processing for Sensor Networks. In Proc. 1st Biennial Conf. on Innovative Data SystRes. (CIDR) 2003. pp. 233-244.

[5] Jianju Chen, David J.DeWitt, Feng Tian, Yuan Wang, A Scalable Continuous Query System for Internet Databases, ACM SIGMOD, 2000.

[6] Babu S. and Widom J., Continuous Queries Over Data Streams, ACM SIGMOD Record archive Volume 30, Issue 3, 2001, 109-120.

[7] Moustafa A.Hammad, Michael J.Franklin, Walid G.Aref, Ahmed K.Elmagarmid, Scheduling for shared window joins over data streams, Proc. VLDB, 2003.

[8] Lukasz Golab M. Tamer Ozsu, Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams, Proc. VLDB, June 2003.

[9] TIMOS K.SELLIS, Multiple-Query Optimization, ACM Transactions on Database Systems, March, 1988.

[10] Yousuke Watanabe, Hiroyuki Kitagawa, A Multiple Continuous Query Optimization Method Based on Query Execution Pattern Analysis, DASFAA 2004.

[11] Hector Garcia-Molina, Jeffrey D.Ullman, Jennifer widom, Database System Implementation, Prentice Hall.

서 영 균

2003년 경북대학교 컴퓨터과학과 학사 2005년 한국과학기술원 전자전산학과 전산학전공 석사. 2005년 3월~현재 한국과학기술정보연구원 NTIS사업단 통합기술개발팀 연구원. 관심분야는 데이타 스트림, 센서 네트워크, XML, 정보 통합, 정보 검색

손 진 현

1996년 서강대학교 전산학과 학사. 1998년 한국과학기술원 전산학과 석사. 2001년 한국과학기술원 전자전산학과 박사 2001년 9월~2002년 8월 한국과학기술원 정보전자연구소 박사후 연구원. 2002년 9월~현재 한양대학교 컴퓨터공학과 조교수. 관심분야는 데이타베이스, e-비즈니스, 임베디드 소프트웨어, 분산시스템

김 명 호

정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조