비트-벡터 해시 테이블을 이용한 효율적인 다중 스트림 조인 알고리즘

(An Efficient M-way Stream Join Algorithm Exploiting a Bit-vector Hash Table)

권 태 형 † 김 현 규 † 이 유 원 † 김 명 호 ††
(Tae Hyung Kwon) (Hyeon Gyu Kim) (Yu Won Lee) (Myoung Ho Kim)

요 약 MJoin은 변화가 잦은 데이타 스트림의 조인을 효율적으로 수행하기 위한 방법으로 소개되었다. MJoin은 다중 스트림의 처리가 가능하도록 대칭적 해시 알고리즘을 확장한 것으로, 각 입력 튜플마다 모든 해시 테이블에 동일한 키를 지닌 튜플이 존재하는지 반복적으로 체크한다. 그러나, 조인 선택율이 낮고 조인되는 데이타 스트림의 수가 많을 경우, 이러한 체크 과정의 성능은 조인되는 데이타 스트림의 조인순서에 많은 영향을 받게 된다. 본 논문에서는 MJoin처럼 대칭적 해시 알고리즘을 기본으로 하지만, 이러한 체크 과정을 조인순서에 상관 없이 상수 시간에 처리하는 BiHT-Join 알고리즘을 제안한다. BiHT-Join은 스트림에 있는 튜플의 존재 유무를 비트-벡터로 유지하며, 이를 비교하는 것으로 조인의 성공/실패를 판단한다. 따라서, BiHT-Join은 이 판단을 기준으로 조인이 성공하는 튜플만 해시 조인을 수행함으로 조인 효율을 높일 수 있다. 우리는 실험을 통해 BiHT-Join이 다중 데이타 스트림 조인에서 MJoin에 비해 더 나은 성능을 제공한다는 것을 보인다.

키워드 : 다중 스트림 조인, 비트-벡터

Abstract MJoin is proposed as an algorithm to join multiple data streams efficiently, whose characteristics are unpredictably changed. It extends a symmetric hash join to handle multiple data streams. Whenever a tuple arrives from a remote stream source, MJoin checks whether all of hash tables have matching tuples. However, when a join involves many data streams with low join selectivity, the performance of this checking process is significantly influenced by the checking order of hash tables. In this paper, we propose a BiHT-Join algorithm which extends MJoin to conduct this checking in a constant time regardless of a join order. BiHT-Join maintains a bit-vector which represents the existence of tuples in streams and decides a successful/unsuccessful join through comparing a bit-vector. Based on the bit-vector comparison, BiHT-Join can conduct a hash join only for successful joining tuples based on this decision. Our experimental results show that the proposed BiHT-Join provides better performance than MJoin in the processing of multiple streams.

Key words : Multi-way stream join, Bit-vector Hash Table

† 학생회원 : 한국과학기술원 전산학과
thkwon@dbserver.kaist.ac.kr
hgkim@dbserver.kaist.ac.kr
ywlee@dbserver.kaist.ac.kr
†† 정 회 원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2007년 9월 11일
심사완료 : 2008년 7월 8일

정보과학회논문지: 데이타베이스 제35권 제4호(2008.8)

1. 서 론

다중 데이타 소스로부터 지속적으로 도착하는 데이타 스트림에 대한 질의 처리는 유비쿼터스 스트림 애플리케이션에 필수적으로 필요하다. 예를 들어, 어느 대형 유통업체에서 수요 예측을 위해 모든 마트에서 공통적으로 잘 팔리는 상품을 실시간으로 모니터링한다고 하자. 관리자는 Q1과 같은 다중 스트림 조인 질의를 수행할 수 있다[1].

```
Q1: SELECT A.ItemName
    FROM Store1 A, Store2 B, Store3 C, Store4 D
```

WHERE A.ItemID = B. ItemID and B. ItemID = C. ItemID and C. ItemID = D. ItemID

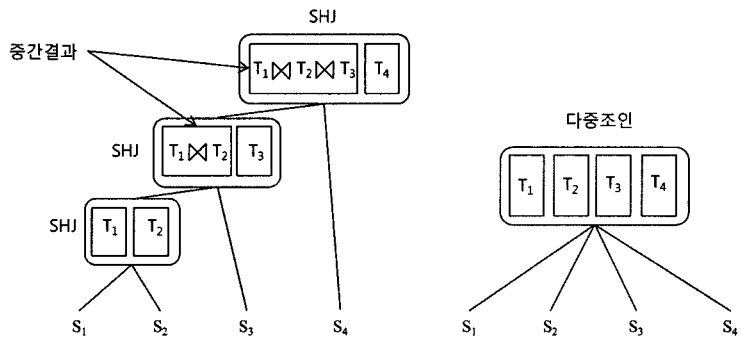
Q1에서 조인되는 데이타 스트림 소스는 n개로 확장될 수 있고, 조인되는 애트리뷰트도 한 개 이상의 짝으로 구성될 수 있다. 이와 같은 질의 예제는 센서 네트워크에서 오브젝트의 경로 파악 및 센서들이 중첩되는 구간에서 중복 탐지되는 물체 처리[2]와 IP네트워크에서 트래픽 모니터링[3], 그리고 주식 거래에서 트랜드 분석[4] 등 다양한 것[5]을 포함한다.

지속적으로 전달되는 다중 스트림 조인을 수행하기 위해, 대칭적 해시 조인(SHJ: Symmetric Hash Join)이 제안되었다[6,7]. 이것은 전체 데이타 소스를 읽기 전에 점증적으로 조인 결과를 생성할 수 있도록 한 바이너리 조인으로, 스트림 조인에 유용하다. 바이너리 조인 연산자(Operator)를 이용하여 다중 조인을 수행하려면, 보편적인 DBMS들에서 하듯이, n개의 스트림을 $(n-1)$개의 SHJ 연산자를 이용하여 파이프라인 형태로 연결하여 조인을 수행한다.

그림 1(a)는 SHJ 연산자를 파이프라인으로 연결한 논리적인 질의 계획(Logical query plan)을 보여 준다. SHJ 연산자는 기본적으로 스트림당 하나의 해시 테이블을 유지한다. S_1에 새로운 튜플이 도착할 경우, 먼저 자신의 해시 테이블 T_1에 해당 튜플을 입력한 후에 S_2의 해시 테이블 T_2를 검사(probe)한다. 이 조인 결과는 $(T_1 \bowtie T_2)$에 보관된다. 다시 S_3의 해시테이블 T_3와 같은 방법으로 조인한 다음, 그 결과를 $(T_1 \bowtie T_2 \bowtie T_3)$에 남기며, 최종적으로 S_4의 해시 테이블 T_4를 검사하여 새로 입력된 튜플에 대한 최종 조인 결과를 출력한다. 만약, S_4에서 새로운 튜플이 입력되면, 중간 결과인 $(T_1 \bowtie T_2 \bowtie T_3)$만을 검사하여 최종 조인 결과를 출력한다. 이러한 조인 방식은 두 가지 제안 사항을 가지고 있다. 첫 번째로 상당한 양의 스트림 카디날리티(Cardinality)를 가정할 때 중간 결과를 유지하는 비용이 크다. 두 번째로 파이프라인 형태로 고정된 조인 실행 트리는 변화가 잦은 스트림의 특성을 감안할 때 적절하지 않다. 왜냐하면, 조인이 진행되는 동안 최적의 질의 계획은 계속적으로 변할 수 있기 때문이다.

다중 스트림 조인에 있어 이러한 문제를 해결하기 위하여, 셋 이상의 데이타 스트림을 하나의 조인 연산자를 사용하여 조인하는 MJoin[8]이 소개되었다. MJoin은 SHJ를 세 개 이상의 스트림에 확장한 것으로 중간 결과들을 유지하지 않는 특징을 가진다. 그림 1(b)는 MJoin의 다중 조인 연산자를 보여 준다. MJoin도 SHJ와 동일하게 스트림당 한 개의 해시테이블을 유지한다. S_1에 새로운 튜플이 도착할 경우, 자신을 제외한 다른 모든 스트림의 해시 테이블을 검사하여 즉각적으로 최종 조인 결과를 출력할 수 있다. 또한, 중간 결과를 유지하지 않기 때문에 튜플 단위로 다른 조인순서를 적용할 수 있다.

그러나, MJoin의 처리율은 조인 순서(Join order)에 많은 영향을 받는다. 예를 들어, 그림 1(b)에서 S_1에서 들어온 입력 튜플이 T_2 및 T_3와는 조인되면서 T_4와는 조인되지 않는다고 하자. 이 경우 T_2부터 T_4까지 순차적으로 검사한다면, T_4까지 3번의 검사를 거친 후 에야 조인 실패를 알 수 있다. 반면, T_4부터 처리한다면 1번의 검사를 통해 조인 실패를 알 수 있다. 하지만, n개의 스트림을 가정할 때, 스트림마다 적절한 조인 순서를 구하는 것은 어려운 문제이다. DBMS에서도 n개의 릴레이션을 조인하는데 있어, 최적의 조인 순서를 구하는 문제는 NP-hard로 알려져 있다[9]. 더욱이, 스트림들은 조인이 진행되는 동안에도 카디널리티와 조인 선택율이 지속적으로 변하는 성질을 가지고 있기 때문에, 최적의 조인 순서 역시 조인 중에 계속 변한다. 따라서, MJoin의 경우 조인 순서 문제에 적절하게 대처하지 못할 경



(a) 파이프라인으로 연결과 SHJ (b) 다중 조인 연산자

그림 1 스트림 다중 조인

우, 불필요한 해시 테이블 검사로 인해 조인 효율이 나빠질 수 있다.

본 논문에서 우리는 불필요한 해시 테이블 검사를 피할 수 있는 새로운 조인 알고리즘 BiHT-join을 제안한다. BiHT-join은 조인의 효율을 높이기 위해 비트-벡터 해시 테이블(*BiHT*: Bit-vector Hash Table)을 유지한다. BiHT에서 각각의 해시 주소는 n개의 비트로 구성된 비트-벡터를 가지며, 이 비트-벡터의 i번째 비트는 i번째 스트림에서 튜플의 존재 유무를 표시한다. BiHT-join은 다음 두 가지 장점을 가진다. 먼저, 제안하는 알고리즘은 MJoin을 기본으로 하지만, 조인 순서 계산이 불필요하다. 따라서, 이를 계산하고 적용하는 부가적인 비용을 제거할 수 있다. 두 번째로 입력되는 튜플의 조인 성공 유무를 BiHT를 이용하여 바로 결정할 수 있다. 이 판단을 근거로 하여 조인이 성공하는 튜플만 실질적인 조인을 수행함으로써 불필요한 해시 테이블 검사 비용을 제거할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 다중 스트림 조인과 관련된 연구를 제시하고, 3장에서는 BiHT-join알고리즘을 MJoin과의 비교를 통해 상세히 설명한다. 4장에서는 비용 추정식을 통해 두 알고리즘을 분석하고, 5장에서는 실험을 통해 BiHT-join의 효율성을 보이며, 6장에서는 결론을 내리도록 한다.

2. 관련연구

일반적으로 전통적인 DBMS에서 다중 릴레이션에 대한 조인은 인덱스나 조인-인덱스[10]를 사용한다. 하지만, 바운드 되지 않고 지속적으로 입력되는 대량의 스트림에 대해서는 이러한 종류의 인덱스를 유지하기는 쉽지 않다. 따라서, 스트림에 대한 조인은 블록(block)하지 않고 조인을 수행할 수 있는 대칭적 해시 조인[6]과 리플조인[11]을 확장한 XJoin[7]이 보편적으로 사용된다. 우리는 다중 스트림을 조인하기 위해, 이 같은 바이너리 조인 연산자를 파이프라인 형태로 연결하여 조인할 수 있다. 하지만, 이 방식은 고정된 질의 수행 계획을 가지고 있기 때문에, 스트림 환경의 변화로 조인 중에 질의 계획을 변경하기 위해서는 중간 결과를 다시 만들어야 하는 추가적인 비용이 발생한다.

MJoin[8]은 이러한 문제점을 개선하기 위해 제안되었다. MJoin은 바이너리 연산자를 확장하여, 다중 스트림을 한번에 조인할 수 있도록 만들어 졌다. 즉, MJoin은 Eddies[12]에서 제안된 바와 같이 튜플 단위로 다른 조인 순서를 적용할 수 있게 되었다. 하지만, 변화가 잦은 스트림 환경에서 최적의 조인 순서를 구하는 것은 NP-hard문제이기 때문에, MJoin만으로 실제 환경에서 조인의 성능을 높이기는 어렵다는 한계가 있다. 또한, MJoin은 중간 결과를 유지 않는 대신, 반복적으로 모든 해시 테이블을 검사하여 최종 조인 결과를 생성한다. 이와 같은 방법은 조인 순서가 올바르게 적용되지 못할 경우 조인 효율을 악화시킬 가능성이 있다.

이 외에도 다중 스트림에 대한 조인 효율을 높이기 위해 여러 가지 연구가 있었다. 대표적인 두 가지는 바로 윈도우의 사용[5]과 다양한 형태의 질의 최적화이다. 윈도우 개념은 바운드 되지 않는 스트림을 제한된 메모리와 컴퓨팅 파워를 통해 처리하기 위해, 시간적/양적 범위로 바운드 시켜 조인하는 것이다. 이 개념은 기존의 여러 알고리즘들에 직접적으로 적용되어 다중 스트림 조인의 성능을 높일 수 있다. 그리고, 후자는 스트림 애플리케이션의 환경을 고려하여 질의 계획을 최적화하는 것으로 크게 입력 스트림의 카디널리티 비율을 고려하여 질의를 최적화하는 방법[13], 스트림의 내용물을 기반으로 질의 계획을 최적화하는 방법[14], 조인 중에 발생하는 중간 결과를 재 사용하는 방법[15]으로 구분 할 수 있다. 이러한 개념과 기법들은 바이너리 연산자 혹은 다중 조인 연산자와 결합하여 다중 스트림 조인의 효율을 높이기 위해 사용된다.

위에서 살펴본 바와 같이 다중 스트림 조인은 가장 많은 비용이 필요한 스트림 연산자 중에 하나이고, 여전히 어려운 난제들을 가지고 있다. 본 논문에서 제안하는 새로운 조인 알고리즘은 MJoin을 근간으로 하지만, 비트-벡터라는 새로운 메모리 구조를 이용하여 복잡한 조인 순서 문제를 탈피하고, MJoin의 장점인 유연성을 유지함으로써 다중 스트림 조인의 성능을 높인다.

3. BiHT-join 알고리즘

3.1절에서 우선 예제를 통해 다중 스트림 조인을 MJoin[8]만으로 처리했을 경우 생길 수 있는 문제점을 지적하고, 제안하는 방식의 해결책을 직관적으로 보여준다. 다음으로 3.2절에서는 메모리 기반 알고리즘을 설명한다. 제안하는 알고리즘은 MJoin을 기반으로 설계되었으므로 MJoin의 기본 알고리즘을 먼저 설명한 후, 제안하는 방식을 비교 설명한다. 3.3절에서 메모리 오버플로우가 발생할 경우 처리할 수 있는 오프라인 알고리즘에 대해여 제안하며, 3.4절에서는 BiHT-join을 사용할 경우 유용한 질의 패턴들을 살펴 본다.

3.1 예제

이번 절에서는 그림 2의 주식 거래 모니터링 예제를 통해, 이것을 MJoin을 이용하여 처리했을 때 발생되는 문제점에 대해 살펴본다. 그림 2에서와 같이 증권사는 세 개의 스트림 데이타 소스 S_1, S_2, S_3로부터 실시간으로 데이타를 받는다고 하자. 외국인, 기관, 개인 모두가 매도하는 상품을 알기 위해, 조인키 '상장사'로 세 개의

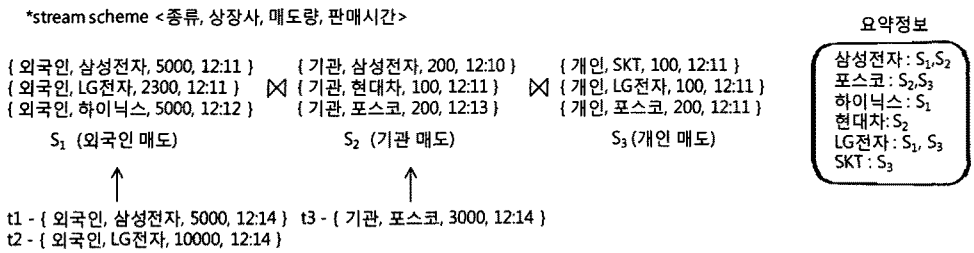


그림 2 증권사에서 매매되는 주식 상품을 모니터링하는 예제

스트림을 조인해야 한다(상단 스트림에 있는 튜플은 이미 조인이 완료된 것이며, 하단 튜플은 조인을 기다리는 튜플이다. t1, t2, t3는 튜플의 도착순서를 가리킨다). 입력 버퍼에 새로 도착한 세 개의 튜플에 대해 MJoin을 적용할 경우, 조인을 진행하기 전에 먼저 조인 순서를 결정해야 한다. 세 개의 스트림에 대한 가능한 조인 순서의 조합은 모두 여섯 가지($S_1 \rightarrow S_2 \rightarrow S_3$, $S_1 \rightarrow S_3 \rightarrow S_2$, $S_2 \rightarrow S_1 \rightarrow S_3$, $S_2 \rightarrow S_3 \rightarrow S_1$, $S_3 \rightarrow S_1 \rightarrow S_2$, $S_3 \rightarrow S_2 \rightarrow S_1$)가 있다. 각각에 대해 C_1, C_2, C_3, C_4, C_5, C_6라 하자. 이 경우 각각의 튜플에 대한 최적의 조인순서는 C_5, C_2, C_1이 된다. 그러나, 실세계 스트림 애플리케이션에서 튜플별로 최적의 조인 순서를 구하여 적용하기는 어렵다. 그리고 n개의 스트림에 대해 최적의 조인 순서를 구하는 문제는 NP-hard문제로 알려져 있다.

MJoin의 문제점을 개선하기 위해 본 논문에서 제안하는 방법은 조인키별로 각 스트림에 해당 키값을 가지는 튜플이 존재하는지를 알려 주는 요약 정보(그림 2의 좌측 요약정보 참조)를 이용한다. 따라서, 새로 입력되는 튜플은 개별 스트림의 메모리 영역을 접근하기 전에, 이 요약 정보를 확인하여 해당 튜플이 조인되는지를 판단한다. 이 판단을 근거로 조인이 성공하는 튜플만 메모리 영역에 보관된 스트림과 조인을 수행한다. 이러한 방식을 사용할 경우 우리는 조인의 실패를 알기 위해 스트림의 메모리 영역에 여러 번 접근하는 비용을 제거할 수 있다.

3.2 기본 알고리즘

MJoin은 입력 스트림마다 해시 테이블을 유지한다. 스트림의 개수를 n이라 하고, 스트림 S_i에 대한 해시 테이블을 $T_i(1 \le i \le n)$라 하자. T_i의 주소는 조인키 k의 해시값에 1대1 대응된다. 여기서, 모든 해시 테이블은 같은 해시함수 $h()$를 사용한다고 가정한다. 그림 3은 세 개의 연속된 프로시져를 가지고 있는 MJoin을 설명한다. 해당 알고리즘은 특정 스트림으로 한 개의 튜플이 도착할 때마다 시작된다.

(1) Hashing : S_i에서 입력된 튜플 r_i의 조인키 k에 대해 $h(r_i.k)$를 통해 해시값 v_k를 생성한다.
(2) Moving : T_i의 v_k주소에 튜플 r_i를 입력한다.
(3) Probing : 임의의 조인 순서로 검사를 수행한다. 이 때, 어떤 $T_j(i \neq j)$에 대한 검사에서 v_k번지에 버킷이 비어 있으면 조인 실패에 해당됨으로, 조인을 중지한다. 그렇지 않을 경우, 최종 조인 결과를 얻기 위해 r_i를 $T_j(v_k)$와 조인한다.

MJoin은 어떤 스트림 소스에서 튜플이 도착하더라도 위와 같은 방법으로 조인을 수행한다. 하지만, 과정(3)에

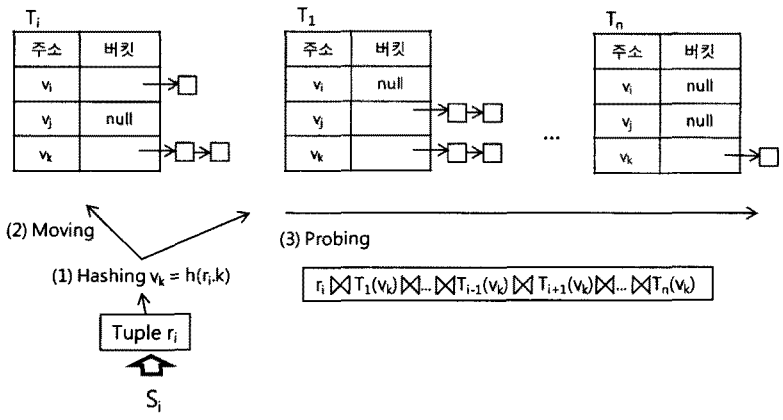


그림 3 MJoin의 조인과정

서 조인 실패에 해당될 경우, 해시 테이블에 대한 검사가 몇 번 수행될지 알 수 없다. 또한, 실패의 경우 과정 (3) 전체가 무의미한 것이 될 수 있다. 조인 실패의 경우에, 각각의 튜플이 최적의 조인 순서에서 처리 되었다면 한번의 검사가 수행될 것이고, 최악의 조인 순서라면 $(n-1)$번의 검사가 수행될 것이다. 즉, 스트림의 개수가 많아질수록 올바른 조인 순서의 선택은 MJoin 성능에 많은 영향을 미치게 되며, 조인 선택율이 낮을 경우 이것의 영향력은 더욱 커질 수 있다.

이러한 문제를 극복하기 위해 BiHT-join은 스트림의 현재 상태를 요약하는 BiHT를 사용한다. 그림 4의 하단에 있는 BiHT는 이것의 예를 보여 준다. 여기서, BiHT와 개별 해시 테이블 T_i는 모두 동일한 해시 함수를 사용한다. 따라서 모든 테이블의 주소 개수는 동일하다. 그림4의 예에서 보면 모든 T_i에는 해시주소 v_k를 갖는 튜플이 존재한다. 따라서, BiHT에서 v_k의 비트-벡터는 모두 1인 '11…1'이다.

비트-벡터에서 i번째 비트의 값: 데이타 스트림 T_i에 해당 해시값을 가지는 튜플이 있으면 1이며, 없으면 0이다.

BiHT-join에서는 입력 튜플에 대해 비트-벡터를 체크함으로써, 해당 튜플이 조인될지 아닐지를 직접적으로 판단할 수 있다. 그림 4는 BiHT-join의 전체 조인 과정을 설명한다. MJoin과의 주요한 차이점은 과정(3)에 있으며 BiHT-join은 입력 튜플이 조인되는 경우에만 해시 테이블에 대한 실질적인 조인 과정을 수행한다.

(1) Hashing : S_i에서 입력된 튜플 r_i의 조인키 k에 대해 $h(r_i.k)$를 통해 해시값 v_k를 생성한다.

(2) Moving : T_i의 v_k주소에 튜플 r_i를 입력한다.

(3) Probing : BiHT의 v_k주소에 i번째 비트를 1로 세팅한다. 그리고, BiHT의 v_k주소에 모든 비트가 1인지 조사하여, 모두 1이면, 최종 조인 결과를 얻기 위해 r_i를 $T_j(v_k)$와 조인한다. 그렇지 않을 경우, 조인 실패에 해당되며, 조인을 중지한다.

BiHT-join에서는 MJoin의 검사 방식과는 다르게 비트-벡터를 비교함으로써 조인의 성공/실패를 결정한다. 따라서, BiHT-join에서는 튜플 당 한번의 BiHT 확인만 수행된다. 이와 같이 조인 성공 여부를 검사함으로써, 우리는 MJoin의 불필요한 해시 테이블 검사 비용을 제거할 수 있다. 하지만, 조인 실패가 드물게 발생할 경우에, BiHT의 유지는 다소 부가적인 작업이 될 수 있다. 이와 관련하여 우리는 4장의 비용 분석과 5장의 실험을 통해 본 논문에서 제안된 알고리즘이 MJoin에 비해 우수한 성능을 보인다는 것을 보인다.

3.3 메모리 오버플로우의 처리

이번 장은 튜플의 입력율(Input rate)을 메모리가 처리할 수 없을 경우 즉, 메모리 오버플로우 상황에서 BiHT-join의 동작에 대해 기술한다. 메모리 오버플로우 문제를 처리하는 기본 원리는 MJoin과 유사하나 비트-벡터를 이용한다는 점에서 다르다. 먼저 MJoin의 처리 방식을 설명한 후, 이를 바탕으로 BiHT-join의 개선점을 설명한다.

MJoin이 메모리 오버플로우에 대처하는 방법은 두 단계로 나뉘어진다. 첫 번째는 디스크 동반 플러시(Coordinated Flush)를 통해 메모리 오버플로우를 처리하는 단계이고, 두 번째는 모든 입력이 중지된 상태에서 플러시된 튜플에 대해 디스크 조인을 수행하는 단계이다. 그림 5는 해시 테이블에 메모리 오버플로우를 처리하기 위하여 플러시 필드가 추가된 T_i와, 각각의 T_i에 대해 디스크로 플러시 될 경우 저장되는 공간인 스트림 파티

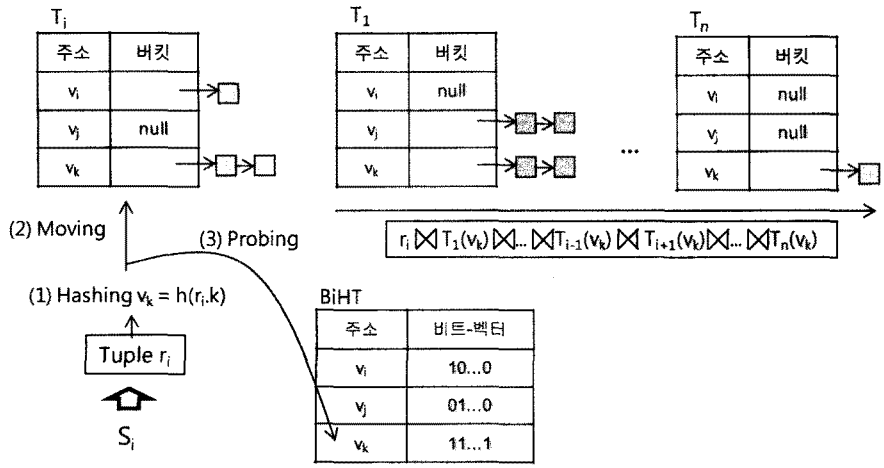


그림 4 BiHT-ioin의 조인과정

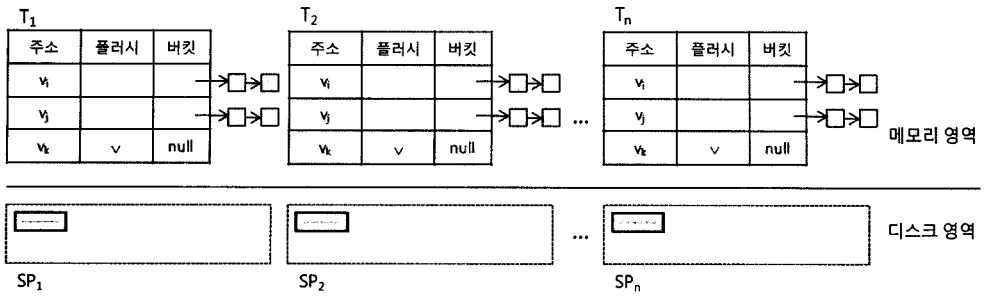


그림 5 MJoin에서 플러시과정

션 SP_i의 구조를 보여 주고 있다. 그림 5의 예로 보면, 해시값 v_k를 가진 모든 튜플은 해당 스트림 파티션에 동반 플러시되었으며, 각각의 T_i의 플러시 필드에 체크 표시가 되어 있는 것을 볼 수 있다.

(1) 메모리 오버플로우 처리 단계

(가) 모든 입력을 중지 시킨다.

(나) 메모리 영역을 확보하기 위해 무작위로 해시값 v_k를 정하여 모든 T_i에서 해시값 v_k를 가진 튜플을 디스크로 플러시하고, T_i에 플러시 상태를 표시한다.

(다) 과정 (나)를 통해 메모리 영역이 확보되면 입력 중지를 해제하고, 메모리 조인을 다시 시작한다. 이때, 그림 5와 같이 플러시 체크가 된 해시값 v_k를 가진 튜플이 입력되면, 조인을 수행하지 않고 곧바로 디스크의 해당 파티션 SP_i에 쓴다. 그렇지 않은 경우 메모리 오버플로우가 발생할 때까지 메모리 조인을 수행한다.

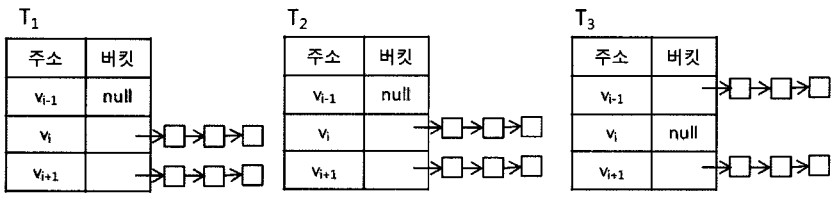
(2) 디스크 처리 단계

이 단계는 모든 입력이 중지된 상태에서 디스크 영역의 스트림 파티션에 대해 디스크 조인을 수행하며, 다시 입력이 재개되면 중단된다.

MJoin은 무작위로 디스크에 플러시될 해시값을 정한다. 하지만, 이 방식은 전체 스트림에서 선정된 해시값을 가진 튜플의 양이 적을 경우 잦은 플러시를 야기할 수 있다. 이 문제는 메모리 오버플로우 상황에서 조인의 효율을 악화시키는 결과를 초래할 수 있다.

이를 개선하기 위해, BiHT-join은 BiHT를 이용하여 플러시될 해시값을 결정함으로써 플러시 횟수를 줄인다. 그림 6은 메모리 오버플로우를 처리하기 위해 플러시 필드가 추가된 BiHT를 보여 준다. 그림 6의 예로 보면, 해시값 v_{i+1}을 가진 튜플을 플러시 할 경우 그렇지 않은 경우에 비해 한 번의 플러시로 더 많은 양의 튜플을 디스크로 보낼 수 있다. 즉, 메모리 오버 플로우 처리 단계에서 우리가 BiHT를 이용하면, 한번의 플러시로 보다 많은 양의 메모리 공간을 확보할 수 있다. 이 경우 플러시 횟수를 줄이는 이득뿐 아니라 더 많은 입력 스트림을 처리함으로써, 메모리 오버플로우 상태에서 조인의 성능을 높인다.

구체적인 알고리즘을 살펴보면 다음과 같다. BiHT-join은 메모리 오버플로우 발생 시 디스크로 플러시할



BiHT

주소	플러시	비트-벡터
v_{i-1}		001
v_i		110
v_{i+1}	v	111

그림 6 BiHT를 이용한 플러시

튜플을 무작위로 선정하는 하는 대신, 여러 개의 해시값을 임의 추출하여 이 중 비트-벡터에 1이 가장 많을 것을 선정함으로써, 한번에 플러시되는 튜플의 양을 많게 한다. 우리는 5장 실험에서 이러한 플러시 기법이 무작위로 해시값을 선정하는 방법에 비해 상대적으로 높은 성능을 발휘함을 보일 것이다.

3.4 유용한 질의 패턴

우리는 BiHT-join이 유용한 여러 개의 질의 패턴을 생각해 볼 수 있다. 이동 물체의 경로를 파악하는 질의의 경우, 질의의 결과로 조인키만 출력하면 된다. 예를 들어 다음과 같은 질의를 생각해 보자. Q2는 세 개의 원격 센서로부터 들어 오는 스트림들을 조인하여 해당 센서를 거치는 오브젝트를 식별하는 질의를 수행한다.

```
Q2: SELECT A.id
    FROM Sensor1 A, Sensor2 B, Sensor3 C
    WHERE A. id = B. id and B. id = C. id
```

Q2의 결과는 개별 스트림의 해시 테이블을 유지할 필요 없이 BiHT만을 유지함으로써 생성할 수 있다. 이와 같이, 질의의 결과로 조인키만 출력하는 경우 MJoin에 비해 BiHT-join은 우수한 성능을 발휘한다. 이러한 예는, 센서들의 중첩 구간에서 중복해서 탐지되는 물체를 식별하는 경우와 네트워크에서 다수의 로컬 사이트에서 동시에 발생하는 침입을 탐지하는 경우 등 다양한 애플리케이션에서 찾아 볼 수 있다.

4. 비용분석

이번 장에서는 비용 추정식을 통해 BiHT-join이 MJoin에 비해 얼마나 우수한지를 설명하고자 한다. 먼저, MJoin은 앞서 3.2절에서 설명한 바와 같이 입력 튜플을 조인하기 위해 세 단계의 프로시저를 수행한다: (1) 입력 튜플을 Hashing, (2) 해당 튜플을 대응하는 해시 테이블에 Moving, (3) 해당 튜플과 유사한 튜플이 다른 튜플에 있는지 검사하고, 그 결과를 출력하는 Probing. 이들 비용을 C_{hash}, C_{move}, C_{probe}로 각각 나타낸다고 할 때, MJoin의 튜플당 조인 비용은 이들의 합으로 나타낼 수 있다. 또한, C_{probe}의 경우, 한 개의 해시 테이블을 검사하는 비용을 C_{comp}라고 하고, 그 평균 검사 횟수를 μ라고 한다면 $C_{comp}\times\mu$로 표현할 수 있다. 주어진 시간에 입력된 전체 튜플을 γ라고 하면, 조인의 전체 비용은 다음 식과 같이 나타낼 수 있다.

$$C_{MJoin}(\gamma, \mu) = \gamma \times (C_{hash} + C_{move} + C_{comp} \times \mu) \quad (1)$$

식 (1)에서, 평균 검사 횟수 μ는 조인 실패 시에 기대값 μ_{fail}과 조인 성공 시의 기대값 μ_{succ}로 나눌 수 있으며 다음과 같이 표현할 수 있다.

$$\mu_{succ} = (n-1) \times o_1 o_2 \cdots o_{n-1} \quad (2)$$

$$\mu_{fail} = 1 \times (1 - o_1) + 2 \times o_1 (1 - o_2) + \cdots + (n-1) \times o_1 o_2 \cdots (1 - o_{n-1}) \quad (3)$$

위 식에서 o_i는 입력 튜플이 주어졌을 때, i번째 해시 테이블에 해당 튜플과 유사한 것이 존재하는가를 나타내는 확률이다. 이 값은 MJoin[11]에서 *selectivity factor*로 설명되어 있다. 이것은 조인 선택율과는 달리, 매치(match) 횟수를 입력 튜플로 나눈 값에 해당되며 1보다 항상 작은 값을 가진다. 식 (2)에서 보듯이 조인이 성공되는 튜플은 $(n-1)$의 검사를 통해 그 결과를 출력하며 그 확률은 전체 o_i의 곱으로 설명될 수 있다.

마찬가지 방법으로 BiHT-join의 조인 비용도 세 개의 프로시져 수행 비용의 합으로 표현된다. 단, C_{probe}는 MJoin과는 달리 $C_{BiHT}+C_{comp}\times\mu_{succ}$로 설명될 수 있다. 여기서, C_{BiHT}는 BiHT를 유지하고 비교하는 비용에 해당되며, C_{comp}는 조인이 성공하는 튜플에만 발생하는 비용이다. 따라서, 조인의 전체 비용은 다음 식으로 표현된다.

$$C_{BiHT\_join}(\gamma, o) = \gamma \times (C_{hash} + C_{move} + C_{BiHT} + C_{comp} \times \mu_{succ}) \quad (4)$$

식 (1)과 식 (4)의 차는 BiHT-join의 우수한 정도를 설명하는 비용 추정식이 되며 다음과 같다.

$$E(\mu_{fail}) = \gamma \times (C_{comp} \times \mu_{fail} - C_{pre}) \quad (5)$$

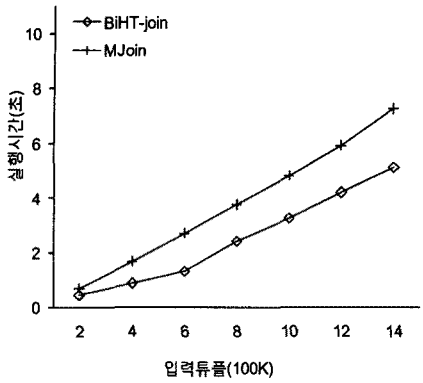
식 (5)에서 E가 양의 값을 가질 때, BiHT-join이 그 양만큼 조인 효율이 높다고 설명할 수 있다. 이 식은 우리의 직관을 그대로 반영하고 있다. 3장에서 설명한 바와 같이 조인 실패를 탐지하는 비용의 차이만큼 BiHT-join의 성능 향상을 기대할 수 있다. 조인 실패가 드물게 발생한다면, BiHT-join의 성능 향상도 그 만큼 감소된다.

5. 실 험

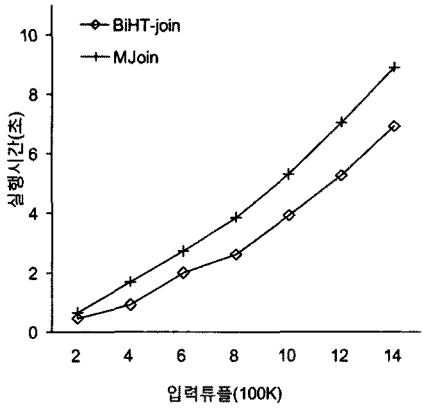
이장에서 우리는 다양한 실험을 통해 MJoin에 비해 BiHT-join이 우수하다는 것을 보이도록 하겠다. 실험은 조인 선택율, 애플리케이션 종류, 그리고 메모리 오버플로우 처리의 세가지 측면을 고려하여 실행되었다. 먼저, 실험을 위해 데이타 생성기를 만들었다. 이것은 입력 스트림의 개수 및 카디널리티와 조인 선택율 등을 파라메터로 하여 가상적으로 스트림을 발생시킨다. 각 스트림의 데이타 집합은 <조인키, 타임스탬프, 기타정보>로 구성된다. 실험은 윈도우 XP환경을 가진 인텔 펜티엄 4.2GHz, 메인메모리 1.5G 장비에서 수행되었다.

조인 선택율에 따른 성능차이

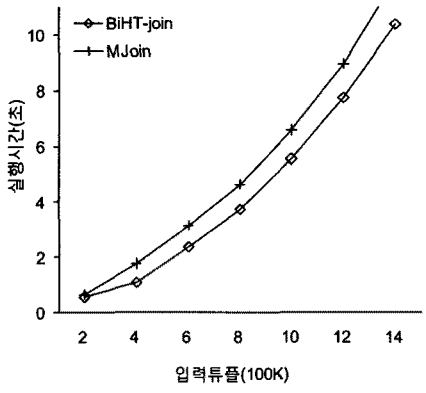
첫번째 실험에서, 두 개의 알고리즘 BiHT-join과 MJoin에 대해 입력 스트림이 세 개인 경우에 조인 선택율을 0.25, 0.5, 1로 늘여 가면서 성능 차이를 측정하였다. 조인 순서에 무관하게 평균 성능을 측정하기 위해, 데이타 생성기에서 튜플의 조인키값은 난수에 의해



(a) 조인 선택율 0.25



(b) 조인 선택율 0.5



(c) 조인 선택율 1

그림 7 3-웨이 조인에서 다양한 조인 선택율을 적용한 실험결과

무작위로 생성하였고, 튜플의 도착도 스트림 순서에 무관하게 무작위로 처리 하였다. 그림 7은 BiHT-join이 MJoin에 비해 대부분의 경우에서 우수한 성능을 가진다는 것을 보인다. BiHT-join이 상대적으로 낮은 조인 선택율에서 MJoin에 비해 좋은 성능을 보인다는 것은 직관적으로 당연하다. 이것은 BiHT-join이 조인 실패를 MJoin보다 빠르게 찾아낼 수 있는 이유에서 기인한다. 즉, 대부분의 입력 스트림이 조인에 실패하기 때문에 우리는 BiHT를 유지함으로써 MJoin의 불필요한 해시 테이블 검사를 제거할 수 있다. 이로 인해 보다 낮은 조인 선택율에서 더 많은 성능 향상을 가져올 수 있었다. 한편으로 조인 선택율이 높다면, 이와 반대로 BiHT를 유지하는 것은 오버헤드로 작용될 수 있다. 하지만, 몇몇 논문[16,17]에서 실제 애플리케이션에서 다중 스트림에 대한 조인 선택율이 상당히 낮다는 것을 보여 준다. 실제로 논문[16]에서는 조인 선택율이 0.1보다 낮은 경우에 대한 실험 결과가 제시되었다. 결과적으로 볼 때 본 논문에서 제안된 BiHT-join은 실용적인 환경에서 높은 성능을 보인다고 말할 수 있다.

BiHT-join에 적합한 질의 패턴

3.4절에서 설명한 바와 같이 BiHT-join이 항상 MJoin에 비해 우수한 성능을 발휘하는 질의 패턴들이 있다. 이것을 명백히 하기 위해, 우리는 3.4절에서 언급한 Q2에 대한 질의 수행 결과를 측정하였다.

이 실험은 입력 스트림이 세 개인 경우에 조인 선택율을 0.5로 설정하여 수행하였으며, 그림 8은 BiHT-join이 MJoin에 비해 3배 이상 높은 성능을 가진다는 것을 보여 준다. 이것은 BiHT-join의 경우, BiHT만 유지함으로써 조인 결과를 생성할 수 있기 때문이다.

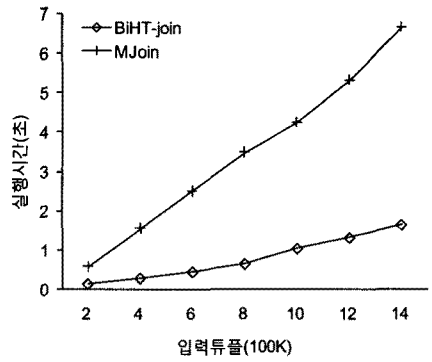


그림 8 3.3장에서 언급한 Q2에 대한 실험결과

메모리 오버플로우 처리

마지막 실험은 3.3절에서 언급한 메모리 오버플로우 상태에서 두 알고리즘의 플러시 방법을 비교 측정하였다. MJoin은 디스크에 플러시할 후보 튜플을 무작위로 선정하지만, BiHT-join은 BiHT에 유지되는 비트-벡터를 근거로 하여 보다 많은 1을 가진 해시값을 그 후보

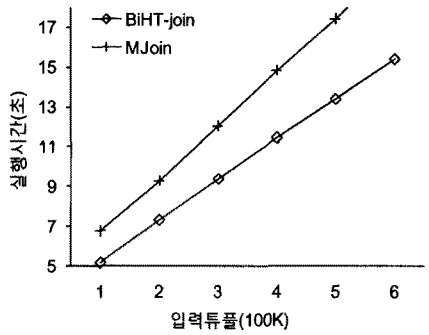


그림 9 메모리 오버플로우 상태에서 3-웨이 조인에 대한 실험결과

로 선정한다. 이 실험을 위해 조인 선택율은 1.0으로 설정되었으며, BiHT-join은 비트-벡터에 1로 설정된 비트가 두 개 이상인 경우에 한하여 플러시하도록 하였다. 그림 9에서 무작위로 플러시 후보를 설정하는 MJoin에 비해 BiHT-join이 메모리 오버플로우 상황에 보다 효율적으로 대처하고 있음을 보여 준다. 이것은 BiHT-join이 MJoin에 비해 한 번에 많은 양의 튜플을 플러시함으로써 전체적인 플러시 횟수를 줄임으로써 달성된다.

6. 결 론

본 논문에서는 다중 스트림 조인을 효율적으로 처리하기 위한 새로운 알고리즘인 BiHT-join을 제안하였다. BiHT-join은 MJoin을 확장한 형태로 설계되었으며, 입력 튜플의 조인 실패 탐지와 메모리 오버플로우를 처리하는 측면에서 MJoin보다 뛰어난 성능을 나타낸다. 또한, 조인의 결과로 조인키만을 출력하는 질의 패턴에 유용할 수 있음을 제안하였다. 이러한 목적을 달성하기 위해 BiHT-join에서는 BiHT(Bit-vector Hash Table)이라는 새로운 데이타 구조를 사용한다. BiHT에서 각각의 해시 주소에는 n개의 비트열로 구성된 비트-벡터가 있으며, 이 값의 i번째 비트는 i번째 스트림에서의 튜플의 존재 유무를 표시한다. 우리는 간단히 이 값을 비교함으로써, 새로 입력되는 튜플이 조인될 것인지를 직접적으로 판단할 수 있고, 이 판단을 기준으로 실질적인 조인을 수행함으로써 조인 성능을 높였다. 끝으로 우리는 다양한 실험을 통해 본 논문에서 제안된 BiHT-join의 효율성을 검증하였다.

참 고 문 헌

[1] Hammad, M.A., Aref, W.G. and Elmagarmid, A.K. (2003): Stream Window Join: Tracking Moving Object in Sensor-Network Database. Proceedings of 15th International Conference on Scientific and Statistical Database Management, Cambridge, Massachusetts, USA: 75-84.

[2] Gehrke, J. and Madden, S. (2004): Query Processing for Sensor Networks. IEEE Pervasive Computing 3(1): 46-55.

[3] Theodore, J., Charles D.C., Oliver S. (2003): Gigascope: A Stream Database for Network Applications. Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, California, USA : 647-651.

[4] Yali, Z., Elke, A.R., and Goerge, T.H. (2004): Dynamic Plan Migration for Continuous Queries Over Data Streams. Proceedings of the ACM SIGMOD international conference on Management of data, Paris, France: 13-18.

[5] Lukas, G. and M Tamer Őzsu. (2003): Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. Proceedings of the 29th international conference on Very large data bases, Berlin, Germany (29): 500-511.

[6] Annita, N.W. and Peter, M.G.A. (1993): DataFlow query execution in parallel main-memory environment. Distributed and Parallel Databases 1(1): 103-128.

[7] Urhan, T. and Franklin, M. (2000): Xjoin: A Reatively-Scheduled Pipelined Join Operator. IEEE Data Engineering Bulletin 23(2): 27-33.

[8] Stratis, D.V., Jeffrey F.N. and Josef, B. (2003): Maximizing the output rate of multi-join queries over streaming information sources. Proceedings of the 29th international conference on Very large data bases, Berlin, Germany (29): 285-296.

[9] Toshihide, I. and Tiko K. (1984): On the optimal nesting order for computing N-relational joins. ACM Transactions on Database Systems (TODS) 9(3): 482-502.

[10] O'Neil, P. and Graefe, G. (1995): Multi-table joins through bitmapped join indices, ACM SIGMOD Record, 24(3): 8-11.

[11] Haas, P. J., Hellerstein J. M. (1999): Ripple Joins for Online Aggregation. Proceedings of the ACM SIGMOD international conference on Management of data, Piladelphia, USA: 287-298.

[12] Avnur, R. and Hellerstein, J. M. (2000): Eddies: Continuously adaptive query processing. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, USA: 261-272.

[13] Viglas, S. and Naughton, J. F. (2002): Rate-Based Query Optimization for Streaming Information Sources. Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 2002, Madison, Wisconsin, USA: 37-48.

[14] Bizarro, P., Babu, S., DeWitt, D. and Widom, J. (2005): Content-based routing: Different plans for different data. Proceedings of the 31st interna-

tional conference on Very large data bases, Trondheim, Norway: 757-768.
[15] Babu, S., Munagala, K., Widom, J. and Motwani, R. (2005): Adaptive Caching for Continuous Queries, Proceedings of the 21st International Conference on Data Engineering, Washington, DC, USA: 118-129.
[16] Hai Y., Ee-Peng L. and Jun Z. (2006): On In-network Synopsis Join Processing for Sensor Networks. Proceedings of the 7th International Conference on Mobile Data Management, Nara, Japan : 32-39.
[17] Yijian B., Haixun, W. and Carlo, Z. (2007): Load Shedding in Classifying Multi-Source Streaming Data: A Bayes Risk Approach. Proceedings of the Seventh SIAM International Conference on Data Mining, Minneapolis, Minnesota, USA: 425-430.

권 태 형
1995년 공군사관학교 전자계산학과(학사) 2002년 국방대학교 전산정보학과(석사) 2006년~현재 한국과학기술원 전산학과 박사과정. 관심분야는 데이타베이스시스템, 센서 네트워크, 스트림 데이타 처리 등

김 현 규
1997년 울산대학교 전산학과 학사. 2000년 울산대학교 전산학과 석사. 2000년~2001년 한국국방연구원 연구원. 2001년~2004년 LG전자 단말연구소 선임연구원. 2005년~현재 한국과학기술원 전산학과 박사과정. 관심분야는 데이타베이스 시스템, 스트림 데이타 처리 등

이 유 원
2005년 부산대학교 전자전기정보컴퓨터공학부 정보컴퓨터공학전공(학사). 2007년 한국과학기술원 전산학과(석사). 2007년~현재 한국과학기술원 전산학과 박사과정. 관심분야는 데이타베이스시스템, 센서 네트워크, 스트림 데이타 처리 등

김 명 호
정보과학회논문지 : 데이타베이스
제 35 권 제 3 호 참조