IE Interfaces
Vol. 16, No. 2, pp. 229-239, June 2003.

실체 뷰의 자기관리에서 완전일관성을 위한 컨베이어 알고리듬

홍인훈 · 김연수†

인천대학교 산업공학과

A Conveyor Algorithm for Complete Consistency of Materialized View in a Self-Maintenance

In-Hoon Hong · Yon Soo Kim

Department of Industrial Engineering, University of Incheon, Incheon, 402-749

The On-Line Analytical Processing (OLAP) tools access data from the data warehouse for complex data analysis, such as multidimensional data analysis, and decision support activities. Current research has lead to new developments in all aspects of data warehousing, however, there are still a number of problems that need to be solved for making data warehousing effective. View maintenance, one of them, is to maintain view in response to updates in source data. Keeping the view consistent with updates to the base relations, however, can be expensive, since it may involve querying external sources where the base relations reside. In order to reduce maintenance costs, it is possible to maintain the views using information that is strictly local to the data warehouse. This process is usually referred to as "self-maintenance of views". A number of algorithm have been proposed for self maintenance of views where they keep some additional information in data warehouse in the form of auxiliary views. But those algorithms did not consider a consistency of materialized views using view self-maintenance. The purpose of this paper is to research consistency problem when self-maintenance of views is implemented. The proposed "conveyor algorithm" will resolved a complete consistency of materialized view using self-maintenance with considering network delay. The rationale for conveyor algorithm and performance characteristics are described in detail.

Keywords: materialized view, view maintenance, self-maintenance, consistency

1. 서 론

정보의 활용을 극대화할 수 있는 새로운 접근 방식인 데이터 웨어하우징(Data Warehousing; DW)이 급부상하고 있다. 데이터 웨어하우징은 기업 내의 정보의 활용 요건을 파악하고, 이를 효과적으로 지원하기 위해 데이터 흐름을 관리하고 운영하는 일련의 활동을 의미한다(조재희, 박성진, 1999). OLAP(On- Line Analytical Processing)와 데이터 마이닝(data mining) 등이 이런 활동들의 종류이다. 그러나 이런 데이터 웨어하우스를 활용하는 데에는 몇 가지 문제점이 있다. 우선 데이터 웨어하우스가 대용량이므로 질의를 처리하는 데 시간이 오래 걸리고, 운영 시스템(operational system)에서 원본 데이터(base data)가 변화하면 데이터 웨어하우스도 이를 반영하기 위해 갱신되어야 한다. 특히 질의를 빠르게 처리하기 위해서 데이터 웨어하우스 내에 실

†연락저자 : 김연수 교수, 402-749 인천시 남구 도화동 177 인천대학교 산업공학과, Fax : 032-761-9952,
e-mail : yonskim@incheon.ac.kr
2002년 9월 접수, 1회 수정 후 2003년 3월 게재 승인

체 뷰(materialized view)를 생성하는데 실체 뷰를 사용함에 따라 두 가지 문제점이 도출된다. 첫째, 실체 뷰를 이용한 질의 재작성방법(실체 뷰의 선택문제)과 둘째, 운영계 시스템에서 데이터가 변화함에 따라 이를 사용하는 실체 뷰도 갱신이 되어야 하는 뷰 관리(view maintenance)방법이다. 뷰 관리는 크게 재계산(recomputation)과 점진적 관리(incremental maintenance)기법 두 가지로 나누어진다. 재계산에 비하여 점진적 관리기법이 효율적이므로 많은 연구들이 점진적 관리기법을 사용하여 알고리듬을 제안하였으며, 점진적 관리기법은 원본 데이터가 존재하는 원천 데이터베이스에 관리질의(maintenance query)를 보내는 Naive Maintenance 방법(Zhuge, Garcia-Molina *et al*., 1995; Zhuge, Wiener *et al*., 1996; Agrawal, Abbadi *et al*., 1997; Zhuge, Wiener *et al*., 1997)과 지역 소스에서 발생하는 각 갱신과 실체 뷰만으로 처리하는 Self-Maintenance 방법(Gupta, Jagadish, Mumick, 1996; Quass, Gupta *et al*., 1996; Huyn, 1997; Samtani, Kumar, 1998; Samtani, Kumar *et al*., 1999), 이 두 개를 혼합한 Hybrid Maintenance 방법(Horng and Chen, 2001)으로 나누어 볼 수 있다. 그간의 많은 연구들은 실체 뷰의 일관성을 유지하기 위하여 Naive Maintenance를 이용하였다. Naive Maintenance는 데이터 웨어하우스와 지역 소스 사이에 통신지연(network delay)과 동시발생(concurrent) 갱신에 의한 이상현상(anomaly)을 제거하기 위해 보상(compensation)을 해야 한다. 따라서 각 알고리듬이 복잡하고 네트워크의 이용률이 높기 때문에 오류가 발생할 경우가 많다. 또한 Self-Maintenance는 실체 뷰를 유지하기 위한 갱신된 데이터가 데이터 웨어하우스 내에 있기 때문에 동시발생 갱신에 의한 간섭(interfering)현상이 발생하지 않게 되어 보상이 필요 없을 뿐더러 Naive Maintenance에서 사용하는 관리질의를 사용하지 않으므로 비일관성 문제를 어느 정도 해결할 수 있다고 간주되었으나 이는 각 갱신 메시지가 역순으로 데이터 웨어하우스에 전달될 수 있는 가능성을 고려하지 않았기 때문이다.

따라서 본 논문에서는 Self-Maintenance를 이용하여 통신지연을 고려한 실체 뷰의 완전일관성(Complete Consistency) 유지 알고리듬을 제안한다. 지금까지 Self-Maintenance에서 실체 뷰의 일관성에서 통신지연 문제에 대한 연구가 없기 때문에 부득이 Naive Maintenance와 비교하도록 한다

2. 기존 실체 뷰 관리 알고리듬의 고찰

2.1 Naive Maintenance

원천 데이터베이스와 데이터 웨어하우스가 분리되면서 전통적인 방법으로 적용한 뷰 관리는 이상현상(anomaly)이 발생한다. 이상현상의 원인은 데이터 웨어하우스와 원천 데이터베이스는 독립적인 시스템으로 존재하기 때문에 원천 데이터베이스가 뷰 정의 및 뷰 유지에 필요한 정보를 알 수 없기 때문이다. 또한 데이터 웨어하우스가 뷰 유지에 필요한 정보를 원천 데이터베이스로부터 얻는 과정에서 원천 데이터베이스의 상태 변화로 정보가 손실 또는 변경이 발생하기 때문에 원천 데이테베이스와의 일관성을 잃게 된다. 초기에 이를 증명하고 이 상현상을 해결하기 위해서 ECA(Eager Compensating Algorithm) 알고리듬(Zhuge, Garcia-Molina *et al*., 1995)이 제시되었으며, ECA 알고리듬은 이상현상을 제거하기 위해 보상질의(compensating query)를 사용하게 된다. 그러나 원천 데이터베이스에서 실체 뷰와 관련된 데이터의 변경이 계속적으로 발생하면 보상에 필요한 보상질의에 따라 메시지의 수가 늘어나는 단점이 있을 뿐만 아니라 원천 데이터베이스에서 데이터의 변경이 정지해야만 실체 뷰가 갱신될 수 있었다. 또한 ECA 알고리듬은 원천 데이터베이스와 데이터 웨어하우스의 개수가 각각 하나일 때만 고려했기 때문에 현실적이지 못하다.

2.2 이상현상(Anomaly)과 보상(Compensation)

ECA(Zhuge, Garcia-Molina *et al*., 1995), Strobe(Zhuge, Wiener *et al*., 1996), SWEEP(Agrawal, Abbadi *et al*., 1997), μ-알고리듬(한근수, 신동천, 1999) 외에 Naive Maintenance 방법을 사용하는 다른 알고리듬들은 필연적으로 발생하는 이상현상을 제거하기 위해 보상질의나 보상처리를 사용함으로써 일관성을 유지시키는 것을 알 수 있다. 결국 Naive Maintenance 방법은 이상현상을 피할 수 없으며 네트워크 상에서 전달되는 관리질의와 결과 집합, 그리고 보상질의로 인한 네트워크 비용이 높아지는 단점을 갖고 있다.

2.3 Self-Maintenance

Naive Maintenance는 원천 데이터가 실체 뷰에서 정의한 기본 릴레이션이 위치하고 있는 각 원천 데이터베이스에만 존재한다는 골격에서 출발한다. 그러나 위에서 설명하였듯이 네트워크의 사용이 증가함에 따라 오류가 발생할 가능성이 있다. 네트워크의 사용량을 줄이고 그에 따른 오류를 줄이기 위해서는 원천 데이터가 데이터 웨어하우스에 위치하는 방법이 있다. 이 방법은 각 원천 데이터베이스에서 발생하는 갱신 메시지가 데이터 웨어하우스에 전달되고 원천 데이터가 데이터 웨어하우스에 이미 존재하기 때문에 관리질의나 보상질의를 원천 데이터베이스에 보낼 필요가 없다. 또한 실체 뷰에 있는 각 튜플들은 이미 기본 릴레이션에 있는 데이터이다. 따라서 원천 데이터베이스에서 발생한 갱신을 실체 뷰에 반영하기 위해 관리질의를 다른 원천 데이터베이스에 보내는 것은 시간소모적인 일이다. 왜냐하면 참조 무결성(referential integrity)을 이용하여 갱신된 데이터와 실체 뷰만으로도 실체 뷰를 최신의 상태로 유지할 수 있기 때문이다. 이를 Self-Maintenance라고 한다.

1) 메시지의 역순(misordering of messages)

통신지연으로 발생하는 이상현상을 보상하기 위해서는 데

이터 웨어하우스에 지연감지 프로세스(DelayDetection)나 이상현상 감지 프로세스(AnomalyDetection)와 같은 별도의 프로세스가 필요하다(한근수, 신동천 1999). 또한 로컬 네트워크에 심한 부하가 걸릴 때는 그 네트워크를 이용하는 전체 메시지의 약 1% 정도에서 통신지연이 발생한다(Chen and Meng, 1997). 실체 뷰의 일관성을 유지하기 위해서는 각 원천 데이터베이스에서 발생하는 갱신 메시지나 데이터 웨어하우스에서 각 원천 데이터베이스에 보내는 관리질의, 그리고 관리질의를 받은 원천 데이터베이스가 데이터 웨어하우스에 전달하는 답변 메시지사이에 순서 있게 처리 되어야 한다. 그러나 데이터 웨어하우스에서 보상처리를 하더라도 오늘날의 복잡한 네트워크 환경에서는 여전히 통신지연이 발생할 수 있다. 따라서 네트워크를 이동하는 각 메시지의 수를 줄일수록 간섭현상이 적게 발생한다고 볼 수 있으므로 실체 뷰를 관리하기 위해 원천 데이터베이스로부터 갱신 메시지만 네트워크를 이용한다면 그만큼 부하를 줄일 수 있다.

2) 메시지 손실(loss of messages)

또한 원천 데이터베이스와 데이터 웨어하우스 사이에서 메시지가 손실될 수 있다. 네트워크에서 패킷(packet)의 손실이 제어 가능하거나 네트워크 층(layer)에서 해결될 수 있다하더라도 네트워크 연결(connection)이 끊어지거나 새로운 연결이 설정되었을 때는 손실된 패킷을 복구할 수 없다. 인터넷 환경에서 네트워크의 부하나 하드웨어 고장으로 인한 이 같은 현상은 보기 드문 문제는 아니다. 이런 메시지의 손실은 각 원천 데이터베이스에서 발생한 갱신이 실체 뷰에 적용이 안 되거나 데이터 웨어하우스가 갱신이 발생한 것을 인지하지 못해서 보상이 수행되지 않을 수 있기 때문에 실체 뷰가 일관성을 유지하지 못할 수 있다(Tok and Eng, 2000).

3) Self-Maintenance의 속성

실체 뷰의 Self-Maintenance를 구현하기 위한 많은 알고리듬들은 실체 뷰 유지에 필요한 부가적인 정보들을 보조 뷰(auxiliary view)를 사용해서 제안하였다. 보조 뷰에는 여러 원천 데이터베이스에 있는 다수의 기본 릴레이션의 튜플 중에서 실체 뷰의 제한조건(where clause)에 만족하나 각 기본 릴레이션 간에 조인(join)이 되지 않은 튜플들이 저장된다. 보조 뷰를 데이터 웨어하우스에 저장함으로써 원천 데이터베이스를 검색할 필요가 없게 되었으나 데이터 웨어하우스에 저장용량 문제가 발생하게 되며 따라서 보조 뷰의 수나 크기를 최소화 하는 것이 문제해결의 실마리가 된다.

Self-Maintenance는 다음과 같은 2가지 속성을 갖고 있다.

Strictly Self-Maintainable: 원천 데이터베이스의 기본 릴레이션 R의 변경사항을 $\triangle R$이라고 했을 때 $\triangle R$과 실체 뷰만으로 실체 뷰의 변경사항 $\triangle MV$를 계산 할 수 있는 실체 뷰의 속성이다. 그러나 이 속성은 $\triangle R$에 있는 튜플과 실체 뷰에 있는 튜플과의 참조(referential)가 없을 경우에는 이를 해결할 수 있는 방법이 없기 때문에 strictly self-maintainability를 보장 할 수 없게 된다. 따라서 해결할 수 있는 방법 중에 하나는 모든 원천 데이터를 데이터 웨어하우스에 복제하는 것이나 이 방법은 저장용량이나 데이터의 중복을 증가시킬 뿐만 아니라 기본 릴레이션의 튜플이 이미 실체 뷰에 존재할 수 있다는 것을 간과하고 있다(Samtani, Kumar *et al*., 1999).

Weakly(generalized) Self-Maintainable: 원천 데이터베이스의 기본 릴레이션 R의 변경사항 $\triangle R$과 실체 뷰, 그리고 보조 뷰를 사용해서 $\triangle MV$를 계산 할 수 있는 실체 뷰의 속성이다(Samtani, Kumar *et al*., 1999).

3. Self-Maintenance의 일관성 문제

먼저 Self-Maintenance에서 일관성의 수준이 어디까지 보장하고 있는지 알아보겠다. <그림 1> 같이 실체 뷰 V의 뷰 트리(view tree)가 있다고 하자.

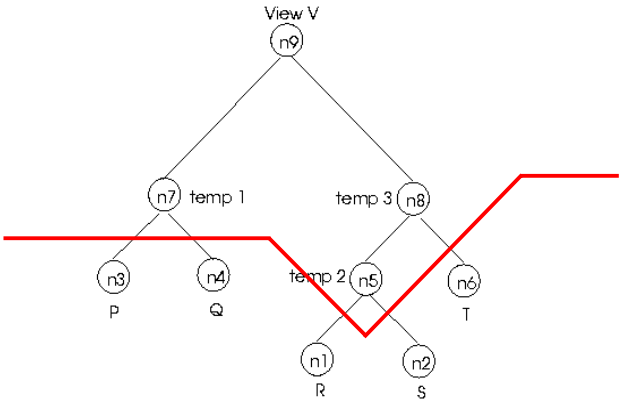


그림 1. 실체 뷰 V의 트리 구조(Samtani, Kumar *et al*., 1999).

실체 뷰 V가 정의한 기본 릴레이션은 P, Q, R, S, T이고 이들 릴레이션은 각기 다른 원천 데이터베이스에 존재하며 중간 노드에 있는 n5, n7, n8은 중간결과 집합(Intermediate Result Sets; IRS)이다. Self-Maintenance 기법에서 중간결과 집합은 실체화가 되지 않고 곧바로 상위 노드로 이동하며 실체 뷰 V를 갱신시킨다. 보조 뷰는 각 중간 노드에 있으면서 조인되지 못한 데이터를 저장하게 된다. 기본 릴레이션 R, S의 데이터 중 조인되지 못한 데이터가 각각 보조 뷰 AV_R, AV_S에 저장되고, T에서 갱신이 발생할 때 각 원천 데이터베이스에서 전송한 데이터가 통신지연으로 인하여 보조 뷰에 늦게 도착하게 되면 각 원천 데이터베이스에서 발생한 갱신을 제때 반영할 수 없게 된다. 따라서 데이터 웨어하우스에서 뷰의 상태변화의 순서가 각 원천 데이터베이스에서 발생한 상태변화의 순서와 다를 수 있으며 원천 데이터베이스 간에 발생한 상태변화 순서가 다르게 반영

될 수 있다. 결국, 모든 실체 뷰는 완전 일관성을 보장할 수 없게 된다.

[보기 1]

위의 뷰 트리에서 실체 뷰 V가 기본 릴레이션 R, S, T만을 정의했다고 가정하자. 각 원천 데이터베이스에 각각 R<A,B>, S<B,C>, T<C,D> 릴레이션이 존재한다고 했을 때 V는 $V(R,S,T)=\Pi_{[A,B,C,D]}((R[A,B]\bowtie_{B=B}S[B,C])\bowtie_{C=C}T[C,D])$로 정의된다. 각 릴레이션의 초기상태가 R<1,2>, S<-,->, T<3,4>이면 AV_R<1,2>, AV_S<-,->, AV_T<3,4>가 된다. 실체 뷰 $V=\varnothing$이다.

U_1=insert(S,<2,3>), U_2=delete(T,<1,2>)가 원천 데이터베이스에서 발생하고 여기에 Self-Maintenance를 적용하자. U_n은 갱신이 발생한 순서를 나타낸다. 그리고 각 원천 데이터베이스에서나 데이터 웨어하우스 내에서는 순차적인 트랜잭션이 가능하고 오직 원천 데이터베이스와 데이터 웨어하우스 사이의 네트워크에서만 통신지연이 가능하다고 하자. 아래 예제는 통신지연이 발생하지 않았을 때를 고려한다.

① 데이터 웨어하우스에서 U_1를 받으면 AV_R<1,2>와 조인된다. 중간결과 집합 MV_{IRS}=<1,2,3>이 되고 AV_T<3,4>와 다시 조인되어 MV_{IRS}=<1,2,3,4>가 된다. 모든 연산이 끝났으므로 실체 뷰 V를 갱신한다. 따라서 V=<1,2,3,4>가 된다. 모든 조인된 AV의 데이터는 삭제된다.

② U_2가 Delete이기 때문에 데이터 웨어하우스에서 U_2를 받으면 실체 뷰 V에서 <1,2>가 있는 모든 튜플을 삭제한다. 따라서 실체 뷰 V는 $V=\varnothing$이 되며 원천 데이터베이스와 완전 일관성을 보이게 된다.

[보기 1]은 통신지연을 고려하지 않은 Strobe 알고리듬에서 사용한 시나리오를 그대로 사용했다. Strobe 알고리듬은 Naive Maintenance이기 때문에 동시발생 갱신이 존재하므로 강한 일관성을 보이게 된다. 또한 C-Strobe는 큐(Queue)를 사용해서 완전 일관성을 보장하나 원천 데이터베이스에서 더 이상의 갱신이 발생하지 않을 때만 실체 뷰가 갱신이 되는 단점이 있다. 따라서 Self-Maintenance를 사용하여 위의 시나리오대로 하면 큐나 그 밖의 다른 프로세스가 없어도 완전 일관성을 보장하게 된다. 그러나 모든 원천 데이터베이스에서 발생한 트랜잭션의 실제 실행순서와 같은 스케줄이 데이터 웨어하우스에 그대로 전달되지 않을 수 있다. 다음은 갱신 메시지의 통신지연으로 순서가 바뀌어 실체 뷰에 일관성 문제가 있음을 보이고 있다.

[보기 2]

각 릴레이션의 초기상태가 R<1,2>, S<2,3>, T<3,4>이고 U_1=insert(R,<5,2>), U_2=insert(T,<3,6>)일 때 나머지 사항은 위의 예와 같다고 하면 AV_R<-,->, AV_S<-,->, AV_T<-,->가 된다. 왜냐하면 각 릴레이션의 모든 초기 상태 튜플이 조인이 되었기 때문에 각 보조 뷰에는 있을 필요가 없다. 실체 뷰 V=<1,2,3,4>이다. 통신지연이 발생하여 U_1=insert(R,<5,2>)보다 U_2=insert(T,<3,6>)가 먼저 데이터 웨어하우스에 도착했을 때 실체 뷰 V는 다음과 같이 갱신된다.

① 데이터 웨어하우스에서 U_2를 받으면 실체 뷰 V가 strictly self-maintainable하기 때문에 U_2는 실체 뷰 V와 조인되어 V는 V={<1,2,3,4>, <1,2,3,6>}이 된다.

② U_1도 실체 뷰 V가 strictly self-maintainable하므로 실체 뷰 V와 조인되어 V는 V={<1,2,3,4>, <1,2,3,6>, <5,2,3,4>, <5,2,3,6>}이 된다.

그러나 실체 뷰 V가 완전일관성이 보장되려면 U_1일 때 V={<1,2,3,4>,<5,2,3,4>}이고 U_2일 때 V={<1,2,3,4>,<5,2,3,4>,<1,2,3,6>,<5,2,3,6>}이 되어야 한다. 따라서 [보기 1]처럼 Self-Maintenance를 이용한 실체 뷰의 완전일관성을 보장해 주기 위해서는 큐나 타임스탬프를 이용해야 한다. 또한 모든 트랜잭션(single update transaction) 스케줄이 실체 뷰에 반영이 된다면 완전일관성을 보여줄 수 있지만 원천 데이터베이스에서 발생한 변경 데이터가 AV에 남아 있게 되면 완전일관성을 보장할 수 없게 된다.

3.1 Key의 활용

일반적인 상황에서 실체 뷰에 정의된 기본 릴레이션의 모든 key 애트리뷰트가 실체 뷰에 있지 않을 수 있다. 따라서 각 기본 릴레이션의 key 애트리뷰트가 보조 뷰나 실체 뷰에 없을 시에는 기본 릴레이션에서 갱신된 튜플을 조인할 수 없다. 참조 무결성 제약조건에 만족하지 않은 튜플을 조인할 수 있는 방법이 없으므로 기본 릴레이션에서 갱신된 튜플은 key 애트리뷰를 포함하여 데이터 웨어하우스로 전송된다. 관계형 데이터베이스는 각 릴레이션의 key 애트리뷰트의 조인연산을 통하여 사용자가 원하는 결과를 얻게 되므로 데이터 웨어하우스에 전송되는 갱신된 튜플이 모든 애트리뷰트를 포함해야 하는지도 문제가 된다. 그러나 실체 뷰가 *Strictly Self-Maintainable*하다면 실체 뷰에 정의된 애트리뷰트와 key 애트리뷰트만이 전송되어도 충분하며, AV를 활용하여 실체 뷰를 관리할 때에는 뷰 트리의 각 노드에 있는 AV가 그 하위 노드의 AV가 갖고 있는 key 애트리뷰트를 포함하고 있어야 한다.

3.2 함수종속(Functional Dependency)

key 애트리뷰트를 사용할 수 없는 상황일 때에는 함수종속을 이용할 수 있다. 함수종속에는 2차 정규형에서 나타나는 부분종속(partial functional dependency)과 3차 정규형에서 볼 수 있는 이행종속(transitive functional dependency)이 있고 실체 뷰와 AV

에서 함수종속이 나타날 경우 key 애트리뷰트를 대신해서 조인을 할 수 있다. 그러나 함수종속을 이용하여 실체 뷰를 갱신할 때에는 각 실체 뷰와 AV에 삽입이상과 삭제이상이 발생할 수 있으며 따라서 이상현상을 제거하기 위해서는 각 튜플을 유일하게 구별할 수 있는 식별자가 필요하게 된다. key 애트리뷰트를 사용할 수 없으므로 각 튜플이 위치한 디스크 위치 값(rowid)을 사용할 수 있다. 오라클(Oracle DBMS)은 모든 행을 유일하게 구별할 수 있는 rowid가 있다. 따라서 중복된 튜플은 서로 다른 rowid를 갖고 있으므로 기본 릴레이션에서 발생한 갱신을 구별할 수 있다. 그러나 물리적으로 지역적으로 분산된 여러 원천 데이터베이스의 rowid의 형식이 버전에 따라 틀려질 수 있으며 이를 관리해야 하는 문제점도 발생하게 된다.

3.3 타임스탬프(Timestamp)를 이용한 일관성

강한 일관성이나 완전일관성을 구현하기 위하여 많은 뷰 관리 알고리듬들은 타임스탬프를 사용했다. 실체 뷰가 정의한 원천 데이터베이스가 물리적으로나 지역적으로 떨어져 있기 때문에 각 원천 데이터베이스에서 발생한 변경사항을 표준시간대를 기준으로 해서 데이터 웨어하우스가 위치한 시간대로 변경할 수 있다. 따라서 각 원천 데이터베이스에서 변경된 데이터를 데이터 웨어하우스에 전송할 때 타임스탬프를 포함하여 전송하며 데이터 웨어하우스에서는 표준시간을 적용하여 이를 변환시켜주면 된다. 전송된 튜플에는 key 애트리뷰트 외에 변경된 시간을 기록할 수 있는 타임스탬프 애트리뷰트가 필요하다.

3.4 큐(Queue)를 이용한 일관성

Strobe Family 알고리듬에는 완전 일관성을 구현하기 위하여 각 원천 데이터베이스에서 발생한 갱신들을 큐에 순서대로 저장하는 C-Strobe 알고리듬이 있다(Zhuge *et al.*, 1996). 물론 각 원천 데이터베이스와 데이터 웨어하우스 사이에 통신지연이 없다는 가정하에 이루어진다. 그러나 통신지연으로 인하여 실체 뷰의 변경 순서가 원천 데이터베이스와 다르다면 완전일관성을 보장할 수 없을 뿐더러 비일관성을 초래할 수 있다. 따라서 통신지연으로 인한 각 원천 데이터베이스의 갱신순서를 정하기 위해서 타임스탬프를 이용할 수 있다. 즉 위의 3.3절에서 설명했듯이 모든 갱신된 튜플의 타임스탬프 정보를 비교하여 시간순서대로 정렬시킬 수 있다.

그러나 단위시간(T_m) 내에 데이터 웨어하우스에 도착한 갱신(U_n)들을 정렬할 수 있다 하더라도 통신지연으로 인하여 늦게 도착한 U_i는 같은 단위시간대에 도착한 다른 갱신들과는 다른 단위시간 T_j에 들어 갈 수 있다($m \neq n$, $i \neq j$). 따라서 단위시간을 얼마만큼 정할 것인지가 문제가 된다. 또한 단위시간 T_j와 T_{j+1}이 있을 때 T_j에는 (U_{i-2}, U_{i-1}, U_i, U_{i+1})이 있고 T_{j+1}에는 (U_{i+2}, U_{i+3},⋯,U_n)이 존재한다고 한다면 U_{i+1}과 U_{i+2}의 순서가 바뀔 수 있다는 문제점이 존재한다. <그림 2>는 이와 같은 현상을 보여준다.

이와 같은 이유로 큐를 이용하여 실체 뷰의 Self-Maintenance의 완전일관성을 구현하는 것은 문제가 발생할 수 있으며 갱신순서를 보정할 수 있는 프로세스가 필요하다.

4. Self-Maintenance의 일관성 유지 알고리듬과 컨베이어 알고리듬

4.1 가정

Self-Maintenance 기법은 관리질의가 필요 없기 때문에 관리질의에 대한 답변 메시지도 필요 없다. 따라서 원천 데이터베이스에서 데이터 웨어하우스로 전송되는 갱신 메시지에서만 통신지연이 발생하며, 이를 보정해 주기만 하면 된다.

가정 1: 실체 뷰는 SPJ(select-project-join)로 정의되며 아래와 같이 표기한다.

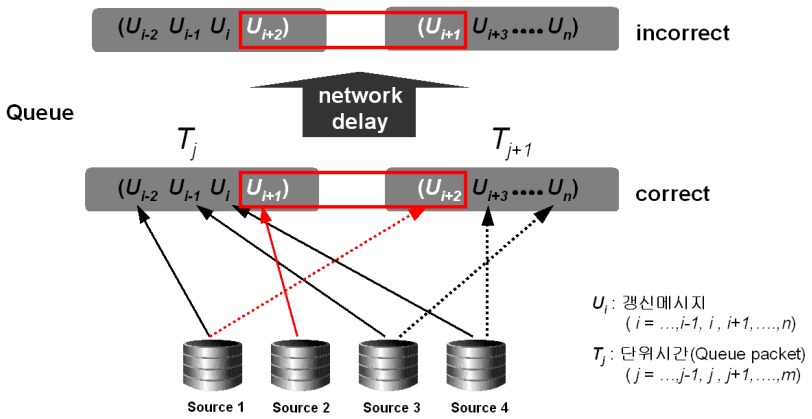


그림 2. 큐(queue) 사이에서 발생한 갱신 메시지의 역순현상.

$$V(R_1,R_2,\cdots,R_N) = \prod_{[proj]}(\sigma_C(R_1 \bowtie_{J.A} R_2 \bowtie \cdots \bowtie_{J.A} R_N)) \quad (1)$$

proj는 실체 뷰의 SELECT 절에 있는 속성들을 나타내며, C는 선택(select)에 해당하는 불리언(boolean) 표현식이다. J.A는 조인 애트리뷰트를 나타낸다.

가정 2: Naive Maintenance 기법의 네트워크 비용보다 Self-Maintenance 기법을 사용하여 데이터 웨어하우스의 저장용량 비용이 적다고 가정한다.

가정 3: 각 원천 데이터베이스는 타임스탬프를 유지할 수 있으며 발생하는 변경사항들은 데이터 웨어하우스에서 유일한 타임스탬프로 유지된다.

가정 4: 실체 뷰와 보조 뷰는 정의된 모든 릴레이션의 key 애트리뷰트가 존재한다. 이는 실체 뷰의 갱신속도를 향상시켜주며 가정 2와 같은 이유를 갖는다.

가정 5: 실체 뷰에 영향을 주는 갱신(relevant update)만 데이터 웨어하우스에 전송한다. 각 원천 데이터베이스에서 발생하는 모든 변경사항을 데이터 웨어하우스에 전송할 필요가 없다. 따라서 실체 뷰의 메타 데이터를 이용하여 원천 데이터베이스에서 실체 뷰에 영향을 주는 갱신만 추출한 뒤 데이터 웨어하우스에 전송하면 네트워크 비용을 낮출 수 있을 뿐만 아니라 원천 데이터베이스의 부하도 줄일 수 있다.

가정 6: 본 논문에서는 Session Consistency를 고려하지 않는다. 이미 상용화된 데이터베이스에서는 트랜잭션 고립화 수준(transaction isolation level)을 결정할 수 있다. 또한 읽기전용 OLAP 질의와 실체 뷰를 갱신하려는 트랜잭션 사이에 발생할 수 있는 락(lock)을 해결하기 위한 멀티버전(multi version) 알고리듬(Kulkarni and Mohania, 1999)이 있지만 본 논문에서는 고려하지 않는다.

4.2 아키텍처

본 논문에서는 WHIPS[1]에서 사용한 아키텍처를 사용한다. WHIPS는 Naive Maintenance 기법을 이용했기 때문에 Self-Maintenance 기법에 맞게 수정하면 <그림 3>과 같다.

WHIPS 시스템 모델과 비교해 보았을 때 위 모델은 WHIPS 시스템 모델의 질의 프로세서와 원천 데이터베이스의 랩퍼를 제거 하였다. 질의 프로세서와 원천 데이터베이스의 랩퍼는

1) WHIPS(WareHouse Information Prototype at Stanford): 데이터 웨어하우스의 생성과 관리를 위한 툴과 알고리듬을 개발하는 프로젝트

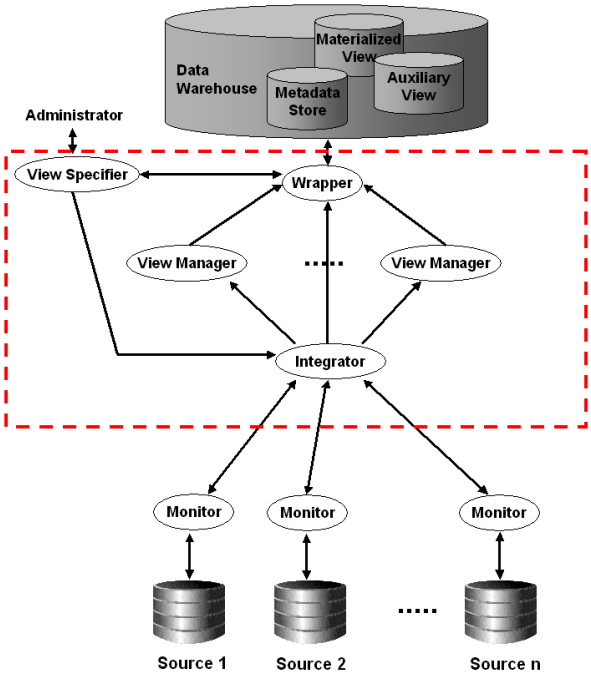


그림 3. 뷰의 자기관리를 위한 시스템 모델.

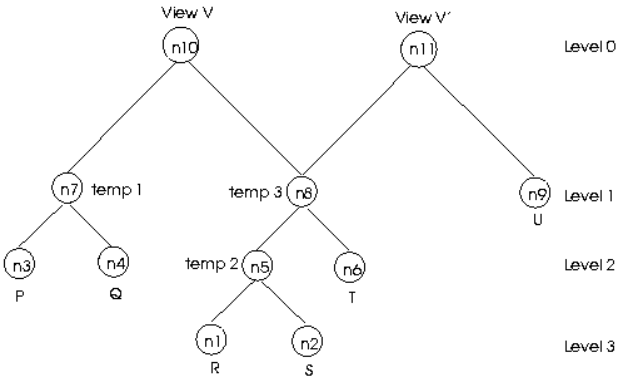


그림 4. 실체 뷰 V and V'를 위한 통합 뷰 트리 구조.

Naive Maintenance 기법을 사용할 때 사용되므로 본 논문에서는 필요가 없다.

① 모니터(monitor): 원천 데이터베이스에서 변경사항이 발생하면 모니터는 이를 감지하여 실체 뷰를 갱신할 수 있는 튜플인지 검토한 후에 적절한(relevant) 튜플을 통합자에 전송한다. 이는 네트워크의 사용량을 최대로 줄이기 위함이다. 적절한 튜플을 추출하기 위해서는 통합자로부터 실체 뷰의 정보를 얻는다.

② 통합자(integrator): 각 원천 데이터베이스의 모니터에서 전송된 튜플을 받아 각 튜플이 영향을 미칠 수 있는 실체 뷰의 뷰 관리자에게 전달한다. 여기에서는 다중 뷰(multiple view)의 뷰 트리(view tree)에서 여러 실체 뷰가 공통의 보조 뷰를 참조할 때 어떤 뷰 관리자를 선택해야 하는지의 문

제가 존재한다. 이것은 보조 뷰 관리정책에 따라 달라질 수 있으며 아래와 같이 3가지로 나누어 볼 수 있다(Samtani, Kumar *et al*., 1999).

<그림 4>에서 노드 n8은 실체 뷰 V와 V'에 의해서 공유된다. 따라서 노드 n8에 존재하는 보조 뷰는 실체 뷰 V와 V'의 뷰 관리자 중 어느 한쪽에 의해서 관리 될 수도 있고 아닐 수도 있다. <그림 5>는 위의 뷰 트리를 각 계층에 있는 부모 노드로 나누어 노드 n8에서 실체 뷰 V와 V'의 보조 뷰 AV^{V}_{tmp3}와 $AV^{V'}_{tmp3}$가 존재함을 쉽게 알 수 있다.

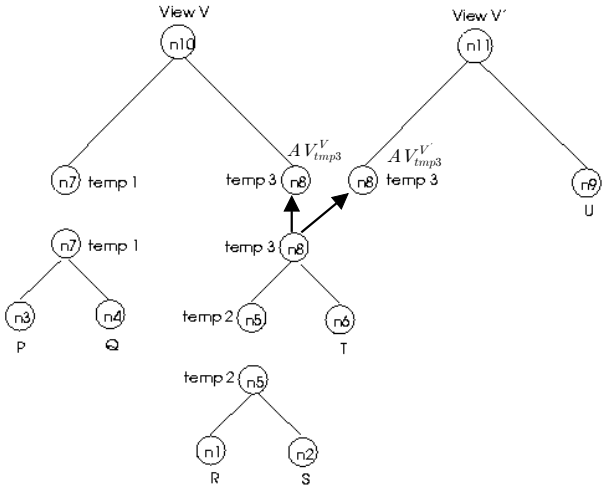


그림 5. 통합 뷰 트리 구조에서의 서브 트리.

원천 데이터베이스에서 발생한 변경사항을 실체 뷰 V와 V'에서 일관성 있게 관리하기 위해서는 n8에서 V와 V'를 위한 보조 뷰 AV_{tmp3}을 생성해야 한다. AV_{tmp3}을 생성하기 위해서는 다음과 같은 3가지 가능성이 있다.

i) Single View Maintenance

V와 V'에 삽입되지 않은 튜플들은 AV^{V}_{tmp3}, $AV^{V'}_{tmp3}$에 각각 저장된다. 그러나 이 구조에서는 $AV^{V}_{tmp3} \cap AV^{V'}_{tmp3}$인 집합이 두 번 계산되며 실체 뷰 V와 V'는 각각 관리된다.

ii) Fully Materialized Multiple View Maintenance

V와 V'에 삽입되지 않은 튜플들은 AV^{V}_{tmp3}, $AV^{V'}_{tmp3}$, $AV^{V\cap}_{tmp3}V$에 각각 저장된다. 따라서 Single View Maintenance에서 발생한 중복을 해결할 수 있으며 계산비용을 절감할 수 있게 된다. 갱신된 튜플이 V와 V'의 제한조건에 모두 만족할 경우 $AV^{V\cap}_{tmp3}V$에서만 조인연산을 하기 때문에 더 효율적이다. 그러나 이 방법은 튜플을 어떤 AV에 접근시켜야 하는지에 대한 부하(overhead)가 발생하게 된다.

iii) Partially Materialized Multiple View Maintenance

V와 V'에 삽입되지 않은 튜플들은 AV_{tmp3} 한 곳에 저장되며 V와 V'에 저장되지 않기 때문에 V와 V'는 부분적으로 실체화가 된다. 즉 V와 V′를 사용하는 질의가 실행될 때에는 부분 실체화된 V와 V'와 AV_{tmp3}을 사용하여 계산한다. V를 사용하는 질의 Q는 다음과 같이 계산될 수 있다.

$$Q=V\cup(AV_{tmp3}\bowtie AV_{tmp1}) \quad (2)$$

또한 V′는

$$Q=V'\cup(AV_{tmp3}\bowtie AV_{U}) \quad (3)$$

이 된다.

이 방법은 질의 재구성(rewrite query)에서 부분 실체화된 뷰와 보조 뷰 AV를 연산하는 것과 같으며, V와 V'를 각 원천 데이터베이스에서 발생하는 변경사항에 따라 일관성 있게 실체화 하지 않기 때문에 뷰 관리비용은 감소하지만 질의 처리비용은 증가하게 된다.

따라서 Self-Maintenance 기법을 이용하여 실체 뷰의 완전일관성을 구현하기 위해서 Partially Materialized Multiple View Maintenance를 사용하는 것은 바람직하지 않다. Single View Maintenance와 Fully Materialized Multiple View Maintenance가 적당하며 두 관리정책 중 통합자는 각 실체 뷰의 정의를 참조하여 뷰 관리자를 선택해야 한다. 그렇게 하기 위해서 통합자는 데이터 웨어하우스 내의 메타 데이터와 상호 교류하여 최신의 실체 뷰 정보를 갖고 있어야 한다.

③ 뷰 관리자(view manager): 각 뷰 관리자는 통합자로부터 넘겨진 튜플을 Self-Maintenance 기법을 이용하여 실체 뷰를 갱신한다. 실체 뷰가 strictly self-maintainable 또는 weakly self-maintainable 인가를 파악하여 관리한다.

④ 데이터 웨어하우스(data warehouse): 메타데이터 저장소, 실체 뷰와 보조 뷰가 존재하며 이들은 랩퍼를 통하여 각 모듈과 통신한다.

⑤ 뷰 지정자(view specifier): 관리자는 이 모듈을 통해서 실체 뷰를 조작하며 통합자와 메타 데이터 저장소에 정보를 전달한다. 통합자는 실시간으로 정보를 전달받을 수 있으며 메타 데이터 저장소에 정보가 저장된다.

4.3 일관성을 위한 조건

앞선 가정에서 모든 실체 뷰는 실체 뷰가 정의한 모든 기본 릴레이션의 key 애트리뷰트를 가진다고 했다. 따라서 기본 릴레이션에서 발생한 삭제 연산은 실체 뷰에 바로 적용할 수 있게 된다. 삽입 연산일 경우는 크게 두 가지로 나누어 볼 수 있다(Samtani, Kumar *et al*., 1999).

① retrievable : 기본 릴레이션에 삽입(insert)된 튜플은 외래 키

(foreign key)를 사용해서 실체 뷰나 보조 뷰 AV에 위치할 수 있다. 실체 뷰에 위치된다는 것은 실체 뷰가 일관성 있게 관리된다는 뜻이며, 삽입된 튜플이 보조 뷰 AV에 위치한다는 것은 실체 뷰의 뷰 트리에서 조인연산을 통해 상위 노드로 이동중에 참조 무결성 제약조건을 만족하지 못해 보조 뷰 AV에 남게 되는 경우를 말한다.

② non-retrievable: 기본 릴레이션에 삽입된 튜플은 실체 뷰나 보조 뷰 AV에 없는 key를 갖고 있다. 즉, 참조 무결성 제약조건을 만족하지 못하므로 삽입된 튜플을 이용하여 실체 뷰를 self-maintenance할 수 없다. 따라서 이런 튜플은 기본 릴레이션의 보조 뷰에 삽입하면 된다. 예를 들면 기본 릴레이션 R에서 non-retrievable 삽입이 발생했을 때 이런 튜플은 AV_R에 삽입하면 된다.

따라서 원천 데이터베이스에서 발생하는 모든 변경사항이 실체 뷰에 적용되지 않을 수 있으며 Naive Maintenance 기법에서도 같은 현상이 발생할 수 있다. 아래 <그림 6>은 기본 릴레이션 R에서 발생한 변경사항이 실체 뷰에 적용될 때 발생할 수 있는 여러 경우를 나타내고 있다.

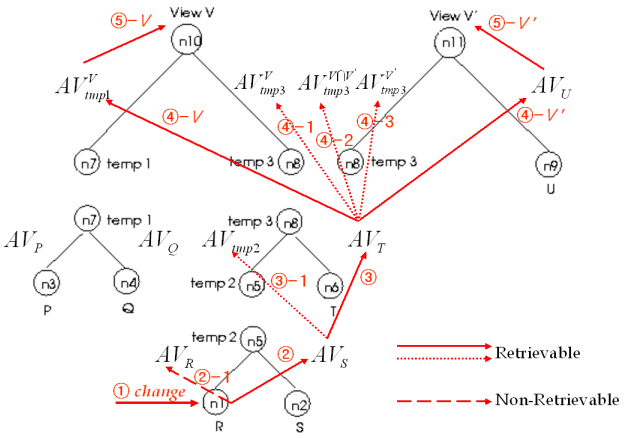


그림 6. 복합 뷰의 자기관리.

위 <그림 6>의 ②-1과 같이 non-retrievable 튜플은 기본 릴레이션 R의 보조 뷰 AV_R에 삽입되고, ③-1과 ④-1,2,3은 retrievable 튜플 중 뷰 트리의 중간 노드 n5, n8에서 조인되지 못하고 보조 뷰 AV_{tmp2}, AV^{V}_{tmp3}, $AV^{V'}_{tmp3}$, $AV^{VV'}_{tmp3}$ V에 저장되는 예를 보여주고 있다. 만일 실체 뷰가 strictly self-maintainable하다면 ①이후에 실체 뷰 V나 V'와 조인연산을 통하여 관리되겠지만, weakly self- maintainable할 경우 ④-V'와 ⑤-V', ④-V와 ⑤-V와 같이 관리되는 것을 볼 수 있다. 이와 같이 실체 뷰에 적용되지 못하고 보조 뷰에 남게 되는 튜플을 UCT(UnConsidered Tuples)라 하자. 따라서 실체 뷰의 완전일관성을 구현하기 위해서는 UCT를 고려하지 않는다. UCT는 참조 무결성 제약조건을 만족하는 튜플이 나타날 때까지 보조 뷰에 저장된다.

4.4 일관성 유지를 위한 컨베이어 알고리듬

각 원천 데이터베이스에서 발생하는 변경순서를 데이터 웨어하우스에 유지시키기 위해서 큐(queue)를 이용하려면 3.4절에서 설명한 것처럼 문제가 발생한다. 그러나 완전 일관성을 유지하기 위해서는 변경 순서대로 실체 뷰에 적용해야 하며 변경이 U_{i-2}, U_{i-1}, U_i, U_{i+1},……,U_n이라고 할 때 U_i는 U_{i+1} 이후에 발생한 변경에 대해서는 고려하지 않아도 된다.

단위시간 T_j와 T_{j+1},……,T_m일 때 각 단위에는 원천 데이터베이스에 발생한 갱신 U_n이 존재하고 갱신들의 타임스탬프를 비교하여 발생 순서대로 정렬된다. 그러나 3.4절에서 설명했듯이 전체 변경을 순서대로 나열하기 위해서는 단위 T_j와 T_{j+1} 사이에 발생하는 오류를 수정해야 한다. 또한 단위 T_j와 T_{j+1} 사이에 발생하는 오류뿐만 아니라 T_j와 T_{j+2} 사이에서 발생할 수 있는 오류도 고려해야 한다. 따라서 통신지연을 검출하는 프로세스 *DelayDetection*은 몇 개의 단위시간으로 검출단위를 결정할 수 있는 매개변수(parameter)가 필요하며 각 단위시간 내에서도 갱신의 개수를 정할 수 있는 매개변수가 필요하다. 단위시간은 갱신의 개수로 나눌 수도 있지만 갱신의 발생빈도에 따라 시간을 기준으로 나눌 수 있다. 따라서 *DelayDetection*은 검출단위 매개변수 *Queues_per_detection*과 갱신개수 매개변수 *Updates_per_queue*로 성능을 향상시킬 수 있다. 그러나 본 논문에서는 전개의 단순함을 위해 이웃하는 단위시간만을 고려하며 각 단위기간에는 4개의 갱신이 있다고 설정한다. 따라서 각 매개변수는 *Queues_per_detection* = 2, *Updates_per_queue* = 4가 된다. 아래 <그림 7>은 *DelayDetection*의 프로세스를 보여주고 있다.

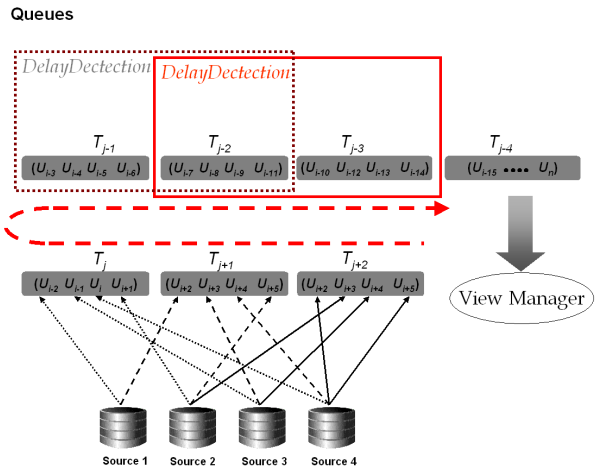


그림 7. 컨베이어 알고리듬(conveyor algorithm).

위의 <그림 7>에서 단위시간 T_{j-4}, T_{j-3}, T_{j-2}, T_{j-1}, T_j, T_{j+1}, T_{j+2}는 컨베이어(conveyor) 방식으로 돌아가며 *DelayDetection*이 두 개의 단위시간을 비교하여 통신지연을 검출한다. T_{j+2}는 현재 발생한 갱신들의 집합이며 T_{j-4}는 검출이 끝난 후 뷰 관리자에 넘겨지는 것을 나타내고 있다. *DelayDetection*은 현재 T_{j-3}, T_{j-2}에서 발생한 통신지연을 검출하여 이를 수정한다. 단위시간 T_{j-3}에

는 (U_{i-7}, U_{i-8}, U_{i-9}, U_{i-11})이 있고 T_{j-2}는 (U_{i-10}, U_{i-12}, U_{i-13}, U_{i-14})이므로 두 단위시간 사이에서 U_{i-11}과 U_{i-10}이 순서가 바뀌었음을 알 수 있다. 따라서 아래 <그림 8>과 같이 수정되어야 한다.

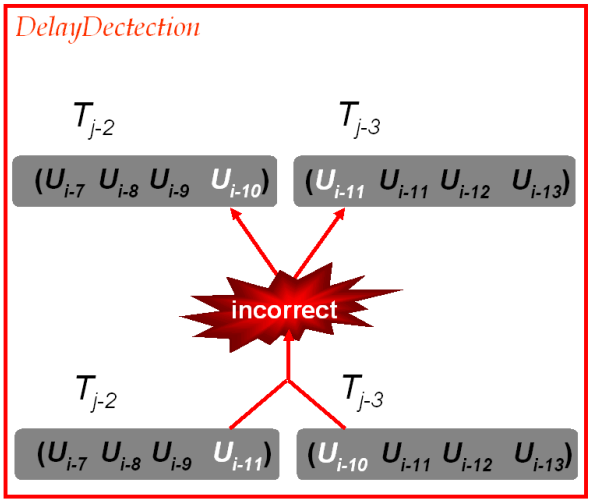


그림 8. *DelayDetection* 프로세스.

그러므로 두 단위시간 T_{j-3}, T_{j-2}는 *DelayDetection*에 의해서 각 갱신의 순서를 바로 잡고 뷰 관리자에 전송된다. 위와 같이 모든 갱신을 정렬한 뒤 뷰 관리자에 보내지는 각 갱신은 실체 뷰가 strictly self-maintainable과 weakly self-maintainable에 따라서 달리 관리된다. 기본 릴레이션 R에서 발생한 갱신 U_i가 있고 U_i는 실체 뷰 V에 영향을 준다고 하자. 또한 실체 뷰 V는 갱신 U_i에 대하여 strictly self-maintainable하면

$$V_{new}=V_{old}\cup(V_{old}\bowtie U_i) \quad (4)$$

과 같이 계산되고 실체 뷰 V가 갱신 U_i에 대하여 weakly self-maintainable하면 다음과 같다.

$$\text{Vnew}=\text{Vold}\cup(A\,V_{tmp1}^{V}\bowtie(\text{AVT}\bowtie(\text{AVS}\bowtie\text{Ui}))) \quad (5)$$

Self-Maintenance 기법을 사용한 실체 뷰는 각 원천 데이터베이스에서 발생한 갱신 메시지를 변경순서에 맞게 데이터 웨어하우스에 적용하면 완전일관성을 보장할 수 있다. 따라서 컨베이어 알고리듬을 이용하여 갱신 메시지의 변경순서를 보장하면 [보기 1]처럼 실체 뷰는 완전일관성으로 관리 된다. <그림 9>는 Self-Maintenance 기법으로 관리되는 실체 뷰의 순서도를 나타내고 있다.

4.3 정당성

데이터 웨어하우스 상태를 ws라 하고 원천 데이터베이스 상태를 ss라고 하면 갱신 U_i에 따른 상태는 각각 ws_i, ss_i가 된다. 또한 뷰 정의 V에 대한 데이터 웨어하우스의 실체 뷰의 상태를 V[ws]로 표현하면 완전일관성을 보장하기 위해서는 $U_i<U_{i+1}$일 때 $V[ws_i] = V[ss_i]$, $V[ws_{i+1}] = V[ss_{i+1}]$, $V[ss_i] < V[ss_{i+1}]$, $V[ws_i] < V[ws_{i+1}]$을 만족해야 한다. 컨베이어 알고리듬으로 모든 갱신은 변경 순서에 따라 나열되어진다. 따라서 $U_i<U_{i+1}$일 때 같은 단위시간 T_m에 U_i와 U_{i+1}가 있다면 뷰 관리자에 의해 차례로 실체 뷰에 적용된다. 따라서 각 갱신에 따라 관리된 실체 뷰는 $V[ws_i] < V[ws_{i+1}]$이 된다. 물론 U_i와 U_{i+1}가 UCT가 아닐 때 가능하며 본 논문에서는 완전일관성을 위해 UCT를 고려하지 않는다. 갱신 U_i에 따른 실체 뷰의 상태가 $V[ss_i]$이면 실제 U_i가 실체 뷰와 조인연산을 통한 상태는 $V[ws_i]$가 된다. 따라서 모든 갱신에 대

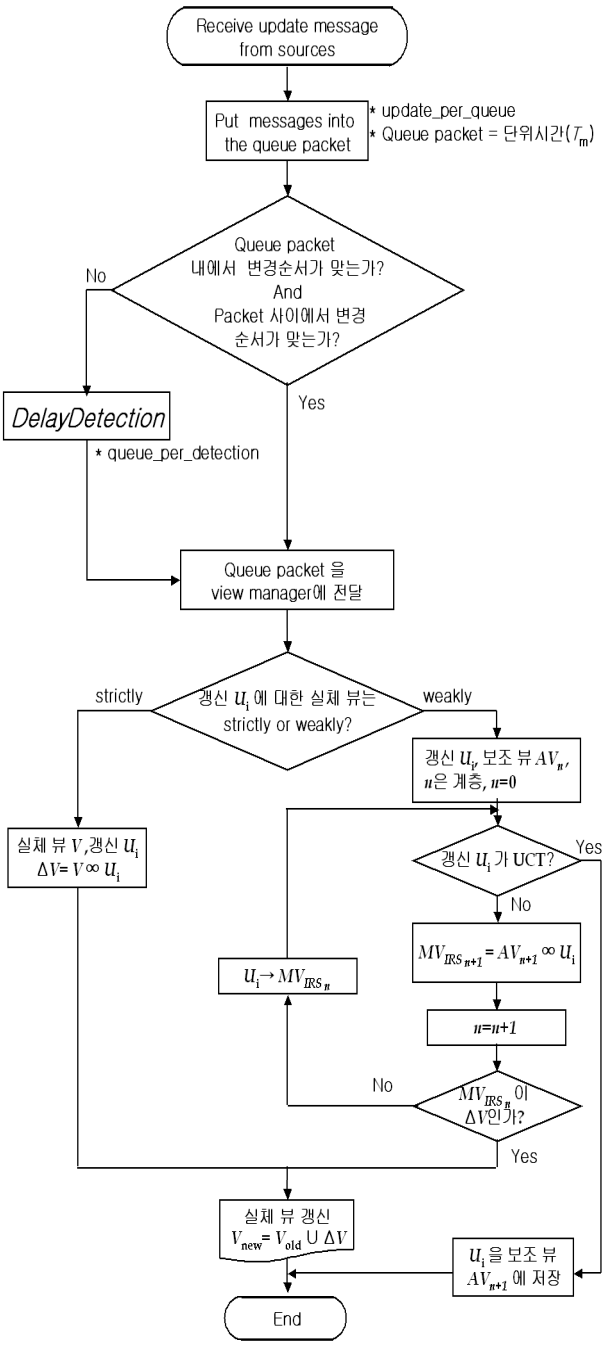


그림 9. 뷰의 완전일관성을 위한 자기관리 순서도.

해서 Self-Maintenance 기법을 이용한 실체 뷰는 각 갱신과 일대일 대응이 되며 완전일관성을 유지한다.

5. 결 론

본 논문에서는 Self-Maintenance 기법을 이용하여 실체 뷰의 완전일관성을 구현하기 위하여 컨베이어 알고리듬을 제안하였다. 대부분의 기존 연구에서는 Naive Maintenance 기법을 사용하여 실체 뷰를 일관성 있게 유지하였고 Naive Maintenance 기법의 특성 때문에 보상에 초점이 맞추어져 있었다. 네트워크의 이용을 줄이기 위해서 보상질의 대신에 보상처리를 통하여 뷰 관리 비용을 줄이려 했으나 이 역시 각 갱신에 따른 관리질의가 필요하기 때문에 네트워크의 사용을 피할 수 없다. Self-Maintenance 기법의 단점은 실체 뷰가 정의한 모든 릴레이션이 데이터 웨어하우스에 존재해야 한다는 것이었는데, 그러나 이것은 실체 뷰에 이미 릴레이션에 있는 튜플(base tuple)이 존재하는 것을 간과하는 것이다. 따라서 Naive Maintenance 기법은 물리적, 지역적으로 분리되어 있는 원천 데이터베이스에 관리질의를 보냄으로써 튜플(base tuple)과의 조인연산을 통한 결과를 다시 전송받는다. 이것은 시간 소모적인 일이며 뷰 갱신속도를 떨어뜨리게 된다. 실제 Self-Maintenance 기법을 이용한 실체 뷰 관리는 Naive Maintenance에 비해 최대 20배 이상 빠르다(Horng and Chen, 2001). 또한 대부분의 논문에서는 바로 이웃하는 갱신, 관리 메시지의 순서만 고려했으나 컨베이어 알고리듬을 이용한 본 논문에서는 그 이상의 통신지연도 검출 할 수 있었다. 다음 <표 1>은 실체 뷰 관리를 위한 각 알고리듬을 비교하였다.

이 논문에서 고려한 UCT는 보조 뷰에 저장되는 유일한 튜플이다. 따라서 Self-Maintenance 기법을 사용하여 실체 뷰를 관리할 때에는 UCT의 발생개수에 따라 저장용량에 영향을 미치므로 UCT의 발생빈도를 조사하여 실체 뷰를 생성해야 한다.

향후 본 논문에서 제시한 컨베이어 알고리듬(Conveyor Algorithm)은 매개변수 *Queues_per_detection*과 *Updates_per_queue*의 상관관계를 계산하여 최적의 실체 뷰 관리가 될 수 있는 연구가 필요하다. 또한 컨베이어 알고리듬이 각 갱신 메시지를 하나씩 처리하는 데 반하여 실체 뷰 갱신속도를 높일 수 있는 병렬처리 방안을 모색한다.

참고문헌

조재희, 박성진, *OLAP 테크놀로지*, SIGMA CONSULTING GROUP, 1999, pp.30~56

한근수, 신동천, 실체 뷰를 기반으로 하는 데이터 웨어하우스 시스템에서 통신지연을 고려한 뷰 유지 알고리듬, *정보과학회논문지*(B), **26**(6), 1999, pp.780~790

A. Gupta, H. V. Jagadish, and I. S. Mumick. Data integration using self-maintainable views, In Proceedings of the 1996 International Conference on Extending Database Technology, March 1996, pp.140~144

D. Agrawal, A. El Abbadi, and A. Singh, Efficient View Maintenance at Data Warehouses, In Proceedings of SIGMOD, 1997, pp.417~427

D. Quass, A. Gupta, I. S. Mumick, and J. Widom, Making Views Self-Maintainable for Data Warehousing. In Proc. Conf. on Parallel and Distributed Information Systems, Miami Beach, FL, Dec. 1996, pp.158~169

JT. Horng, CW. Chen, A mechanism for view consistency in a data warehousing system, *Journal of Systems & Software*, **56**(1), 2001, pp.23~37

N. Huyn, Multiple-View Self-Maintenance in Data Warehousing Environments, *The VLDB Journal*, 1997, pp.26~35

R. Chen, and W. Meng, Precise Detection and Proper Handling of View Maintenance Anomalies in a Multidatabase Environment, CS-TR-97-01, Department of Computer Science, State University of New York at Binghamton, 1997 http://citeseer.nj.nec.com/chen97precise.html

S. Kulkarni and M. Mohania, Concurrent Maintenance of Views Using Multiple Versions, In proceedings of International Database Engineering and Applications Symposium, 1999, pp.254~259

S. Samtani, V. Kumar, Maintaining Consistency in Partially Self-Maintainable Views at the Data Warehouse, Proceedings of the Ninth International Workshop on Database and Expert Systems Applications, 1998, pp.206~211

S. Samtani, V. Kumar, Mukesh K. Mohania, Self Maintenance of Multiple Views in Data Warehousing, In proceedings of ACM International Conference on Information and Knowledge Management, 1999, pp.292-299.

Tok Wang Ling, Eng Koon Sze, A model for evaluating materialized view maintenance algorithms, Web Information Systems Engineering, 2000. Proceedings of the First International Conference on , Volume: 1, 2000, pp.374~382

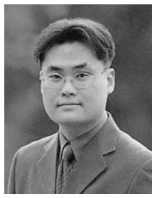
Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom, View Maintenance in a Warehousing Environment, SIGMOD 95, San Jose, California, June 1995, pp.316~327

Yue Zhuge, H. Garcia Molina, Janet L. Wiener, The Strobe Algorithms for Multi-Source Warehouse Consistency, In Proceedings of Fourth International Conference on Parallel and Distributed Information Systems, 1996, pp.146~157

Y. Zhuge, J. L. Wiener and H. Garcia-Molina, Multiple View Consistency for Data Warehousing, In Proceedings of the International Conference on Data Engineering(Binghamton, UK), 1997, pp.289~300

표 1. 뷰 관리 알고리듬의 비교

algorithm	maintenance method	source type	compensation type	network delay	level of consistency
ECA	Naive	single	query	not considered	strong
Strobe	Naive	multi	local	not considered	strong
C-Strobe	Naive	multi	query	not considered	complete
SWEEP	Naive	multi	local	not considered	complete
μ-algorithm	Naive	multi	local	considered	complete
Conveyor	Self	multi	-	considered	complete



홍 인 훈
인천대학교 산업공학과 학사
현재: 인천대학교 산업공학과 석사 과정
관심분야: 데이터 모델링, 데이터 웨어하우징, OLAP, 데이터베이스 튜닝

김 연 수
숭실대학교 산업공학과 학사
New Mexico State University 산업공학과 석사
Texas Tech 산업공학과 박사
현재: 인천대학교 산업공학과 교수
관심분야: 정보시스템, 데이터베이스, 신뢰성 공학