제19회 한국정보처리학회 춘계학술발표대회 논문집 제10권 제1호 (2003.5)

웨어하우스 환경하에서 데이터 통합에 관한 연구

이현창*
*경인여자대학 전산정보
e-mail:hclee@kic.ac.kr

A Study on Data Integration in a Warehouse Environment

Hyun-Chang Lee*
*Dept of Computer Information, Kyung-In Weman College

요 약

데이터 웨어하우스 시스템은 사용자에게 다양하고 고품질의 정보 서비스를 제공하며, 의사 결정을 지언하는데 빠른 질의 처리 요구에 적합한 시스템이다. 고품질의 정보 서비스를 제공하기 위해서는 축적된 많은 정보가 요구되며, 이들 데이터들에 대한 분석을 수행함으로써 경영자의 의사 결정에 최적의 정보를 추출하여 제공해주는 시스템이다. 이러한 의사 결정을 위한 기존의 관계형 데이터베이스 환경하에서는 많은 시간적인 낭비 요소가 존재한다. 이에 본 논문에서는 데이터 통합을 보다 향상시킬 수 있는 전략을 제시하고 기존에 알려진 방법들과 이에 대한 비교 분석을 통하여 향상된 결과에 대해 살펴본다.

1. 서론

현재 상태의 데이터뿐만 아니라 과거의 데이터까지 유지함으로써 이들 누적된 통합 데이터를 분석하여 필요한 정보를 추출할 수 있는 최적의 시스템 구성 형태가 데이터 웨어하우스이다. 일반적인 경우에 데이터 웨어하우스에는 하나의 큰 사실 테이블을 갖고 있으며, 사실 테이블이 분석 등을 수행하기 위해서 다른 여러 테이블들과 조인해야 한다. 이때 조인해야할 질의가 수만, 수 십만개의 레코드를 집계하는 경우 처리시간이 상당히 길어지고 질의 처리 성능 저하를 초래 할 수 있다. 특히, 질의 처리시에 소스에서 가져온 데이터가 소스와 일치하지 않게 되면 사용자에게 부정확한 데이터를 제공함으로써 의사 결정에 큰 영향을 미칠 수 있다.

이에 따라, 소스와 데이터 웨어하우스 사이에서 데이터 정확성을 유지시키기 위해서 실체 뷰가 중요하게 다루어지고 있는 실정이다.[2] 뷰에 대한 대부분의 연구는 뷰 생성에 사용된 소스 테이블에 변경이 일어날 때 실체 뷰를 점진적으로 변경하여 소스와 데이터 웨어하우스 사이에 데이터를 정확하게 유지시키는 기법이다.[3]

일반적으로 뷰에 대한 개념은 주로 뷰 관리가 소스와 웨어하우스 사이에 독립적으로 수행되고 있다는 사실이다. 이러한 환경은 소스가 웨어하우스에 대한 정보를 알 수 없으며, 웨어하우스도 소스에서 변경된 데이터에 대해 인식할 수 없다는 것이다.[1] 이로 인하여 소스에서 갱신이 발생하여 데이터 변경이 발생하게 되면 독립적으로 운용되는 웨어하우스에서는 소스에서 발생된 갱신 정보를 알지 못하기 때문에 갱신정보가 뷰에 바로 적용될 수 없게 된다. 이로 인하여 데이터 웨어하우스는 부정확한 뷰의 원인이 되며, 사용자는 정확한 분석 정보를 수행할 수 없게 된다. 그러므로 소스와 웨어하우스 사이에 일치된 최신의 정보를 제공하고 관리 해야만하는 책임이 발생하게 된다[4]

* 본 연구는 2002년도 경인여자대학 교내연구비 지원에 의하여 수행되었음.

본 논문의 구성은 다음과 같다. 제 2장에서는 본 논문과 관련된 기존 연구들에 대하여 살펴보며, 제 3장에서는 소스와 웨어하우스 사이에 일관된 데이터를 유지시키기 위한 방법에 관하여 살펴본다. 제 4장에서는 간략하게 비교 평가를 수행한 결과에 관해 살펴보며, 마지막으로 결론을 맺는다.

2. 관련 연구

소스와 웨어하우스 사이에 최신의 정보를 유지하기 위해서 소스는 발생한 갱신 정보를 웨어하우스 측에 알려야 한다. 웨어하우스에서는 소스에서 보내온 정보가 뷰를 유지하는 다른 관련 테이블들과 죠인 관계성을 알기 위해서 다시 소스에 질의를 보내어 관련성 여부를 살펴볼 필요가 있게 된다. 소스에서는 웨어하우스에서 보내온 질의가 뷰와 관련된 다른 소스 테이블들과 관련성이 존재하는지 검사해 보아야 한다. 상기 과정이 평가 단계이며, 평가된 결과는 뷰 유지를 위해서 웨어하우스에 보내게 될 내용이다.

이와 같은 방법을 기반으로 이루어진 점진적인 뷰 유지 알고리즘들에 대해서 뷰 유지 시기와 소스 데이터 분산에 따른 뷰 유지 알고리즘의 분류를 살펴보면 다음과 같다.

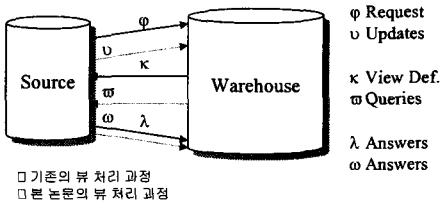
첫째, 즉각적인 뷰 유지 알고리즘 방법이다[1,5] 즉각적인 뷰 유지에 관한 연구는 소스에서 하나의 갱신이 발생하게 되면 그즉시 뷰를 갱신하는 방법이다. 둘째, 첫째 방법에 관한 연구에서 단점은 소스에서 갱신이 발생할 때마다 항상 웨어하우스에 최신 뷰 유지를 위한 연산을 수행하여야 한다.

이로 인하여 이전에 연산이 수행된 데이터들에 대해서도 재계산이 이루어지는 시간과 자원 낭비를 초래하는 단점이 존재한다. 특히, [1]에서의 문제는 소스에서 갱신 정보를 받은 웨어하우스 시스템은 갱신 정보에 따른 질의 관리를 모든 응답 메시지가 올 때까지 관리하게 되어 의사 결정 지원 시스템으로서 충분한 역할을 수행하지 못하는 오버헤드를 초래한다.

뿐만아니라, 모든 갱신 연산에 대해서 웨어하우스 시스템에 갱신 정보가 전달되어야만 하는 문제를 포함하고 있다. 이는 소스와 웨어하우스 사이에서 전달되는 메시지 양을 증가시키는 원인이 되며, 웨어하우스와의 관련성 파악을 위한 오버헤드를 유발한다. 그 이외에 복잡하게 이루어지는 보상 알고리즘 체계이다. 이는 쉽게 관리가 이루어질 수 있는 알고리즘이 필요함을 의미한다.

이러한 단점을 보완하기 위해서 질의가 뷰에 제기될 때 뷰를 갱신하는 지연된 뷰 갱신 연구들이 제시되었다.[4] 본 논문에서는 두 번째 연구를 배경으로 이루어졌으며, 분산 데이터 시스템 환경으로 확대 적용할 수 있지만 단일 환경에서 이루어지는 뷰 유지 방법에 관하여 살펴본다.

먼저, 본 논문에서는 소스에서 발생된 갱신 정보는 뷰 정보를 얻기 위해서 단순 메시지 정보인 요구 정보만을 웨어하우스에 보낸다. 웨어하우스에서는 소스에 뷰 정보를 보내며, 뷰 정보를 받은 소스는 동시성을 유지하면서 평가를 수행하여 그 결과를 웨어하우스에 보낸다. 소스로부터 결과를 받은 웨어하우스는 결과를 적용하여 최신의 뷰 상태를 유지할 수 있게 된다.



<그림 1> 소스 갱신 처리 단계

3. 데이터 일관성을 위한 데이터 통합 전략

본 장에서는 소스에서 발생된 갱신 정보를 웨어하우스에 빠르게 최신 정보로 유지할 수 있는 알고리즘에 대해서 살펴본다. 먼저, 본 알고리즘을 위한 사건(event)은 [3]에서처럼 소스와 웨어하우스에서 발생하는 사건으로 나누며, 뷰 유지를 위해 소스와 웨어하우스에서 유지되는 알고리즘은 [1,3]에 나타나 있다. 본 논문의 소스와 웨어하우스에서 발생하는 사건을 살펴보면 다음과 같다.

3.1 소스/웨어하우스의 사건들

소스와 웨어하우스 사건들은 원자성을 가지며, 하나의 사건내에서는 다음에 기술된 순서대로 수행이 이루어진다고 가정한다.

1) 소스에서의 사건

- S_reqi : 발생한 갱신 정보 Ui가 뷰 정보와 관련성이 존재하는지 확인하기 위해 순서리스트에 등록한후 소스에서 웨어하우스로 뷰 정보 요청 신호를 발생한다.
- S_chki : 웨어하우스로부터 Wvie 정보를 받아서

소스에서 발생한 Ui가 뷰의 정의에 서나타난 테이블을 갱신하는지 검사한다.

- S_evai : 순서리스트 내에서 갱신정보 Ui보다 먼저 존재하는 Uj(j<i)중에서 단지 Ui와 죠인할 테이블이 존재하는지를 평가한다.
- S_exei : 발생한 갱신정보 Ui를 수행하며, 순서리스트에서 Ui를 삭제한다.
- S_sedi : 소스에서 응답 테이블 Ai를 웨어하우스에 보낸다.

2) 웨어하우스에서의 사건

- W_viei : 소스에서 보내온 뷰 정보 요청 신호(Sreqi)를 받고서 단지 웨어하우스의 뷰 정보만을 소스에 보낸다.
- W_ansi : 소스에서 보내온 응답 테이블 Ai를 받아서 Ai를 기반으로 뷰를 갱신한다.

상기 사건들 중 소스 발생 사건이 보상 알고리즘(ECA) 보다 많은 이유는 동시성을 증가시키며, 현실성을 반영하기 위해서 하나의 사건을 세부적으로 나누었다. 갱신 정보 발생은 동시성을 위해서 모두 순서 리스트에 저장된다. 또한 본 논문에서 사용되는 뷰는 [1]에서와 같이 정의하였다.

$$V = \prod_{proj} (\sigma_{cond} (r_1 \times r_2 \times \ldots \times r_n))$$

proj는 애트리뷰트 이름들의 집합이며, cond는 불린(boolean) 수식이며, r1, r2, ... , rn은 서로다른 테이블이다.

3.2 뷰 유지 알고리즘

순서리스트를 이용한 뷰 유지 알고리즘은 [3]에서 다루어졌던 소스/웨어하우스 사건 발생에 따른 점진적 알고리즘을 바탕으로 이루어졌다. 본 논문에서의 갱신 처리는 다음과 같다. 소스에서 하나의 갱신(Ui)이 발생하게되면 Ui는 순서리스트에 저장되며, Sreqi를 웨어하우스에 보내고 다시 웨어하우스에서 보내온 Wviei 정보를 순서에 관계없이 받는다. Ui는 Wvie와 관련성이 존재하는지를 검사하며, 뷰에 연산이 필요한 Ui라면 [3]에서처럼 갱신 질의(Ui)를 뷰에 대해 평가한. 이때 순서리스트를 검사하여 Ui 연산과 죠인하게될 테이블에 대한 갱신 정보 Uj(j<i)가 Ui보다 앞쪽에 존재하는지 검사한다.

만약, Ui연산과 관련된 죠인 테이블이 존재하지 않는다면 갱신 정보 Ui는 소스에있는 테이블에 갱신 연산을 수행하게된다. 수행이 끝나게되면 Ui 정보는 순서리스트에서 삭제되며, 응답 테이블 Ai가 소스에서 웨어하우스로 보내진다. Ai 정보를 받은 웨어하우스는 뷰에 적용함으로서 점진적인 뷰 유지가 이루어지도록 할 수 있다. 또한 한 테이블에 대해 발생하는 많은 갱신 연산은 한 테이블에 대해 수행되므로 순서리스트내의 갱신 정보의 순서에 관계없이 수행 가능하다는 것이다.

위와 같은 결과를 얻기위해 순서리스트, SALUS를 사용하며, SALUS는 소스에서 발생한 모든 갱신 정보를 발생 순서대로 포함하며 순서화(serializability)를 유지함으로서 정확한 결과를 얻을수 있다. 또한 여러 테이블에 대해 갱신 연산의 결과가 정확하게 유지될 수 있어야한다. 소

4. 성능 평가

본 논문에서의 뷰 유지 방법은 기본적으로 [3]의 알고리즘을 따르지만, 소스와 웨어하우스의 역할을 변경하여 성능 향상을 도모하였다. 성능 평가의 정당성을 위해서 본 논문에서는 [1]과 같은 평가 항목을 적용한 환경에서 연구가 이루어졌다.

성능 평가 비교 대상은 전체 뷰의 재계산 알고리즘(RV)과 보상 알고리즘(ECA), 그리고 본 논문에서 제시한 뷰 유지 알고리즘을 비교하였다. 재계산 알고리즘(RV)은 처음부터 모두 검색하여 뷰를 재계산하는 반면에, 보상 알고리즘 ECA는 점진적인 갱신 알고리즘이기 때문에 RV보다 우수한 성능 향상을 기대할 수 있다. 그러나 ECA는 점진적인 뷰 유지에 필요한 더많은 질의가 필요하다. 마찬가지로 ECA에서는 모든 갱신 정보를 웨어하우스에 보냄으로서 큰 갱신 정보 메시지가 전송되며, 뷰 유지를 위해서도 복잡한 보상 알고리즘이 적용되어야한다.

4.1 성능 평가를 위한 변수들

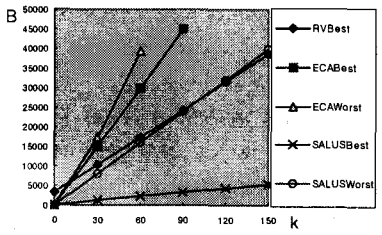
성능 분석을 위해 성능 평가에 사용되는 변수들을 <표 1>에 나타내었으며, 자세한 성능 평가 방법은 [1]에 나타나있다. 본 논문의 성능 평가절에서는 기존의 논문들과 다른 사항들만을 언급한다. RNS는 TS의 구성 요소인 애트리뷰트 가운데 문자열 속성값을 이용하였다.

다음 <그림 2>에서는 갱신 횟수와 전송된 바이트 수의 관계를 그림으로 나타내었다.

<그림 2>는 크기가 작은 테이블로서 카디널러티가 100개인 테이블로서 [1]과 같은 환경으로 평가를 하였으며, RV Worst의 성능은 너무 많은 차이를 보이기 때문에 그림에서 생략하였다.

Name	Meaning	Default Value
M	Number of messages sent	N/A
B	Total Number of bytes transfered	N/A
IO	Number of I/O's	N/A
C	Cardinality of a relation	100
S	Size of projected attributes	4 bytes
σ	Selection factor	1/2
J	Join factor	4
K	# of tuples per physical block	20
k	# of updates at the source	N/A
s-1	# of updates skipped before recomputing view	s≤k
RNS	Size of relation name	52 bytes
TS	Size of Table schema	182 bytes

<표 1> 변수 목록



<그림 2> 갱신 회수와 전송 바이트수 관계

<그림 2>에서 ECAWorst, ECABest에서 일련의 갱신 정보 수가 11과 13보다 적을 때 RVBest 보다 좋은 성능을 보였지만 제한된 갱신 정보 수에 따라서 성능의 차이를 보인다는 사실을 알 수 있다.
그러나 SALUS를 이용한 알고리즘에서 최악의 경우에도 89개보다 적을때 좋은 성능을 보였지만 최상의 알고리즘에서는 갱신 정보의 수가 증가함에따라 오히려 더 좋은 성능의 차이를 보인다는 사실을 알 수있다.

5. 결론 및 향후 연구

데이터 웨어하우스는 의사 결정에 필요한 정보를 하나의 통합된 시스템 환경으로 구축하여 효과적인 질의 응답이 이루어지도록 한다. 효율적인 질의 응답을 지원하기 위해서 웨어하우스는 실체 뷰를 가지고 있다. 본 논문에서는 실체 뷰를 유지하는 방법 가운데 중앙 집중식 알고리즘을 제시하였으며, 다음과 같은 장점을 갖는다.
첫째, 웨어하우스와 관련된 갱신 정보만 송신함으로서 정보 전달에 따른 오버헤드를 줄일수 있다. 둘째, 갱신된 정보의 접근 테이블 수에 따라 성능의 향상을 얻을 수 있다. 즉, 하나의 테이블에 대한 삽입·삭제 갱신 정보들에 대해서는 처리 순서에 관계없이 수행이 이루어져 동시성을 증가시킬 수 있다. 셋째, 복잡한 보상 알고리즘보다 보다 쉽게 관리가 가능한 SALUS를 사용하였다. 넷째, 뷰와 관련성이 없는 갱신 정보에 대해서는 소스에서 뷰 정보와 평가 과정만 거치게 되면 기존 트랜잭션 처리 시간과 비슷한 성능을 가진다.
향후 연구로서 중앙 집중된 환경의 실체 뷰 유지에서 분산된 환경으로 확장과 관리가 용이한 실체 뷰 유지 알고리즘을 연구가 필요하다.

참고문헌

[1] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. Proc. of the ACM SIGMOD Conference, pages 316-327, San Jose, California, May 1995.

[2] Latha S.Colby, Akira Kawaguchi, Daniel F.Lieuwen, Inderpal Singh Mumick and Kenneth A. Ross. Supporting Multiple View Maintenance Policies. In Proceedings of the ACM SIGMOD International Conference on Management of Data pages 405-416. 1997.

[3] J. A. Blakeley, P. Larson, and F. W. Tompa. Efficiently Updating Materialized Vews. Proceedings of ACM SIGMOD 1986 International Conference, pages 61-71, Washington, D.C., May 1986.

[4] A. Gupta and I. S. Mumick. Maintenance of Materalized Views: Problems, Techniques, and Applications. IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2):3-19, June 1995.

[5] T. Griffin and L. Libkin. Incremental Maintenance of Views with Duplicates. In Proc. of ACM SIGMOD 1995.