논문 00-25-8A-21 한국통신학회논문지 '00-8 Vol.25 No.8A

데이터 웨어하우스에서 점진적 뷰 유지를 위한 효율적인 알고리즘

준회원 이 현 창*, 정회원 김 충 석**, 김 경 창*

An Efficient Algorithm for Incremental View Maintenance In a Data Warehouse

Hyun-Chang Lee*, Chung-Seok Kim**, Kyung-Chang Kim* *Regular Members*

요 약

데이터 웨어하우스는 사용자의 의사 결정에 필요한 정보를 제공하여 효율적인 데이터 마이닝 질의 처리 및 그에 대한 응답을 이루도록 한다. 이를 위해서 데이터 웨어하우스는 소스 데이터로부터 유도된(derived) 실체 뷰를 저장하고 있다. 특히, 소스 데이터가 단일 소스 환경에서 잘 알려진 보상 알고리즘을 들 수 있다. 보상 알고리즘에서는 질의 평가 결과를 얻기 위해서 뷰와 관련된 갱신 발생이 많을수록 웨어하우스의 복잡성과 메시지 양이 증가하며, 웨어하우스 내에 질의 관리 오버헤드가 발생되는 문제점이 있었다. 본 논문에서는 뷰 유지를 위한 질의 관리 오버헤드를 감소시키며 정확성을 향상시킨 알고리즘을 제시한다. 또한 메시지 전송과 데이터 전송 측면에서 제시하고 있는 알고리즘을 보상 알고리즘 및 재 계산 알고리즘과 성능을 분석 비교하였다.

ABSTRACT

A data warehouse is able to accomodate efficient data mining query processing and subsequent response by providing information needed for decision making. In such an environment, the data warehouse stores materialized view derived from various sources to enhance query processing. The compensating algorithm to maintain materialized view is well known for a single source site environment. In the compensating algorithm, several problems arise to get results in view maintenance. The problems are due to the overhead in query management within the data warehouse, increased complexity to manage queries in the warehouse as updates occur and increased volume of message traffic between the data source and the warehouse. In this paper, we propose a new algorithm that reduces the overhead in managing queries for new maintenance and that enhances the correctness. We also measured the performance of the new algorithm by evaluating the performance of the existing recomputing view and compensating algorithm and comparing the results with the proposed algorithm.

Ⅰ. 서 론

데이터 웨어하우스는 의사 결정이나 데이터 마이닝과 같은 분석을 목적으로 수행하는데 필요한 데이터 즉, 주제 지향적(subject oriented)이며, 통합적(integrated)이며, 시간 변화(time varying)등의 요구에 적절히 대처할 수 있도록 구성된 데이터 집합인

* 홍익대학교 전자계산학과(kckim@cs.hongik.ac.kr),
** 신라대학교 컴퓨터 정보공학부(cskim@silla.ac.kr)
논문번호 : 00038-0128, 접수일자 : 2000년 1월 28일
※ 본 연구는 한국과학재단 특정기초연구의(과제번호:97-0102-04-01-3) 지원으로 수행되었습니다.

뷰(view)로써 기존의 온라인 트랜잭션 시스템 환경에서 소요되는 로딩 시간을 감소시키는데 사용된다[sweep]. 근래의 연구에서는 데이터 웨어하우스 데이터 집합인 실체 뷰(materialized view)가 더욱 중요하게 다루어지고 있으며[2] 나아가 웹 환경에서도 데이터 웨어하우스 기술이 그대로 적용될 수 있도록 웹 웨어하우징이 출현하게 되었다[9]. 뷰에 대한 대부분의 연구는 뷰 생성에 사용된 소스 테이블에 변경이 일어날 때 실체 뷰를 점진적으로 변경하는 기법이다[5]. 뷰에 관한 기존의 연구 방법에서는 주로 각 소스에서 뷰에 관한 관리 방법을 알고있으며, 질의에 따른 뷰와의 관련성을 알고 있다는 가정하에서 출발한다. 그러나 데이터 웨어하우스 환경에서 소스는 뷰에 관한 정보를 알지못한다[1]. 이로 인하여 소스의 변경이 뷰에 바로 적용될 수 없으며, 이것은 데이터 웨어하우스에서 부정확한 뷰의 원인이 된다. 그러므로 뷰와 소스를 일관성있는 뷰로 유지할 필요가 있다[7].

일반적으로 갱신 정보에 대해 뷰와 소스사이에 이루어지는 관리는 몇 단계를 거치게 된다[1,5]. 먼저, 정확한 뷰 테이블을 유지하기 위해서 소스 측에서 발생한 갱신 정보를 웨어하우스 측에 보낸다. 둘째, 웨어하우스에서는 소스에서 보내온 정보가 뷰를 유지하는 다른 관련 테이블들과의 죠인 관계성을 알기 위해서 다시 소스에 질의를 보내어 관련성 여부를 살펴볼 필요가 있게 된다. 셋째, 소스에서는 웨어하우스에서 보내온 질의를 받아서 현재의 기본 릴레이션들을 사용하여 평가한 후, 평가 결과를 웨어하우스에 반환한다. 이에 관한 사항을 그림 1에 도시하였다.

그림 1을 기반으로 본 논문에서 제시하는 갱신 처리 모델은 분산 환경의 데이터 웨어하우스 환경으로 확장할 수 있다. 또한 본 논문에서 우리는 단일 소스 환경에서 실체 뷰 유지에 관한 몇 가지 알고리즘을 보이고 이들 알고리즘에서 존재하는 문제점들을 줄이기 위한 새로운 알고리즘을 제시한다.

본 논문은 다음과 같이 구성되었다. 제 2장에서는 관련연구를 살펴보며, 제 3장에서는 본 논문에서 제시하고있는 뷰 유지 알고리즘에 대해 설명한다. 제 4 장에서는 본 논문에서 제시하고 있는 알고리즘의 성능 평가를 다른 알고리즘들과 비교하였으며, 제 5 장에서 결론 및 향후 연구 방향으로 본 논문을 구성하였다.

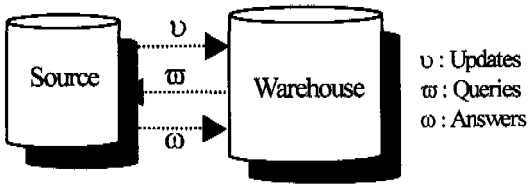

그림 1. 단일 소스 모델에서 갱신 처리 단계

Ⅱ. 관련 연구

뷰 유지 시기와 소스 데이터의 분산에 따른 뷰 유지 알고리즘의 분류를 살펴보면 다음과 같다. 대부분의 점진적인 뷰 유지 알고리즘들은 소스 시스템에 존재하는 테이블이 갱신된 이후에 뷰를 갱신하는 즉각적(immediate) 뷰 유지 알고리즘들이었다[1,4,5]. 이와는 대조적으로 [3,6]의 알고리즘들은 질의가 뷰에 대해 제기될 때만 뷰를 갱신하는 지연된(deferred) 뷰 갱신 알고리즘들이다. 소스 데이터가 분산된 분산 시스템 환경하에서 뷰 유지 알고리즘 연구는 [8]을 들 수 있으며, 단일 소스 데이터 환경은 [1]을 들 수 있다.

다음은 뷰 유지 관리를 위해서 사용된 관련 알고리즘이며, 본 논문에서 제시된 알고리즘과 성능 비교를 위한 알고리즘들이다. 첫째, 뷰의 재계산(recompute the view, RV) 알고리즘으로서 뷰의 재계산이 발생하는 시점은 소스의 갱신 발생과 함께 전체 뷰를 다시 계산하는 방법이다. 본 방법에서는 시간과 자원 낭비를 초래한다는 단점이 존재한다. 보상 알고리즘[1]은 전체 뷰를 다시 계산할때 발생하는 오버헤드를 피하기 위한 알고리즘으로서 테이블 갱신이 즉각적으로 이루어지는 알고리즘이다. ECA 알고리즘은 데이터가 단일 소스 환경에 적용되는 알고리즘이다.

단일 소스 데이터 환경의 보상 알고리즘인 ECA[1]을 기반으로한 뷰 관리는 다음과 같은 문제점이 존재한다. 첫째, 소스에서 갱신 정보를 받은 웨어하우스 시스템은 갱신 정보에 따른 질의 관리를 모든 응답 메시지가 올 때까지 관리하게되어 의사 결정 지원 시스템의 본래의 기능에 충실하지 못한다. 더욱이 모든 응답 메시지를 받을 때까지 unanswered query set을 관리해야만 한다. 둘째, 모든 갱신 연산 정보는 뷰 정보와 관계없이 데이터 소스로부터 웨어하우스로 전송된다. 이는 죠인 테이블의 확인 없이 메시지 전송이 이루어지므로 소스와 웨어하우스 사이에서 전달되는 메시지 양을 증가시키는 원인이 되며, 웨어하우스와 관련성 파악을

위한 오버헤드를 유발한다. 이와같은 문제점을 해결하기 위해 본 논문에서는 효율적인 뷰 유지 알고리즘을 제시한다.

Ⅲ. 점진적 뷰 유지를 위한 효율적인 알고리즘

본 장에서는 점진적인 뷰 유지를 위한 효율적인 알고리즘에 대해서 살펴보기 전에 소스에서 발생하는 갱신은 튜플 삽입 혹은 삭제 같은 하나의 튜플 액션으로 구성되었다고 가정한다. 첫 번째 할 일은 소스와 웨어하우스에서 발생하는 사건(event)들에 대해 살펴보며, 소스에서 행위가 웨어하우스의 뷰와 다르게 일어나는 환경에서 정확성에 대한 정의를 하고자 한다. 다음으로 효율적인 뷰 유지 알고리즘에 대해 살펴본다. 소단원에 관한 내용을 간단히 살펴본다.

1. 소스/웨어하우스의 사건들

소스와 웨어하우스의 사건들은 원자성을 가지며 같은 사이트에서 수행되는 일련의 연산에 해당한다. 하나의 사건 내에서는 상기 기술된 순서대로 수행이 이루어지며, 서로 다른 사건들 사이의 순서는 무관하게 수행될 수 있다고 가정한다.

1) 소스에서의 사건

- S_reqi : 발생한 갱신 정보 Ui가 뷰 정보와 관련성이 존재하는지 확인하기 위해 SALUS에 등록한후 소스에서 웨어하우스로 뷰 정보 요청 신호를 발생한다.
- S_chki : 웨어하우스로부터 Wvie 정보를 받아서 소스에서 발생한 Ui가 뷰의 정의에서 나타난 테이블을 갱신하는지 검사한다.
- S_evai : SALUS 내에서 갱신정보 Ui보다 먼저 존재하는 갱신 Uj(j<i)중에서 단지 Ui와 조인할 테이블이 존재하는지를 평가한다.
- S_exei : 발생한 갱신정보 Ui를 수행하며, SALUS에서 Ui를 삭제후 결과를 전송한다.

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 사건들중 소스 발생사건이 기존의 알고리즘들 보다 많은 이유는 보다 동시성을 증가시키며, 현실성을 반영하기 위해서 하나의 사건을 세부적으로 나누었다. 갱신 정보 발생은 동시성을 위해서 모두 SALUS라는 순서 리스트에 저장된다. 또한 본 논문에서 사용되는 뷰는 다음과 같이 정의하였다[1].

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

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

2. SALUS를 이용한 뷰 유지 알고리즘

본 절에서는 사용자들의 접근시 순서화(serializability)를 유지하기 위해서 SALUS(Simple Algorithm using a List for an Update Serializability)를 이용한 뷰 유지 알고리즘에 대해서 살펴본다. SALUS에서 뷰를 유지하기 위한 기본적인 단계는 다음과 같다. 데이터 소스에서 발생된 갱신의 데이터 웨어하우스에 대한 요청은 단지 뷰 정보만을 얻기 위해 웨어하우스로 보내진다. 데이터 소스로부터 요청을 받은 웨어하우스는 질의가 아닌 단지 뷰 정보만을 데이터 소스로 보낸다. 뷰 정보를 받은 데이터 소스는 뷰와 연관된 즉, 웨어하우스를 구성하고 있는 테이블들과 조인이 이루어지고 있는지 아닌지를 결정하기 위해서 질의를 평가한다. 평가가 이루어지고서 질의에 상응되는 결과 대답이 생성되고 이 결과는 웨어하우스로 보내진다. 데이터 소스로부터 결과를 받은 웨어하우스는 뷰를 갱신하여 일관된 상태를 유지하기 위해 실체 뷰에 적용한다. 그림 2는 위에서 언급한 단계를 도시하고 있다.

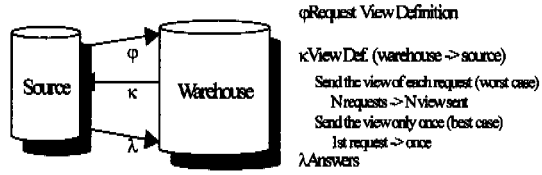

그림 2. 단일 소스 환경에서 SALUS를 이용한 뷰 갱신 처리

본 논문에서 사용되는 뷰는 다음과 같이 정의하였다. $V = \prod_{proj} (\sigma_{cond} (r_1 \times r_2 \times \cdots \times r_n))$

proj는 애트리뷰트 이름들의 집합이며, cond는 불린(boolean) 수식이며, r1, r2, ... , rn은 서로다른 테이블이다. 그림 3는 점진적인 뷰 구성 알고리즘을 도시하고 있다.

```
MODULE Data Source;
  SALUS : initially ∅; /* update list  */
  Tname  : STRING;  /* name of table*/
  ΔR, Answer : RELATION;
  NumSALUS : INTEGER initially 0;
              /* number of updates */
BEGIN  /* start source processes */
  PROCESS(RequestView);
  PROCESS(CheckJoin);
  PROCESS(Evaluate);
  PROCESS(Execute);
END Data Source.
```

```
FUNCTION Check(ViewDef : list of join
Tables; NumRelation : INTEGER) : INTEGER
  VAR
    i, flag : INTEGER;
  BEGIN
    i, flag = 0;
    FOR(i=0; i < NumRelation ; i++) DO
      IF(Tname = ViewDef[i])
      THEN flag = 1; ENDIF;
    ENDFOR;
    RETURN(flag);
END Check;
```

```
PROCESS RequestView;
  BEGIN
    LOOP
        RECEIVE ΔR FROM R;
        APPEND ΔR TO SALUS;
      Tname = table name of ΔR accessing;
        INCREASE NumSALUS;
        SEND Request TO Data Warehouse;
    FOREVER;
  END RequestView;
```

```
FUNCTION Compare(ViewDef: list of join
Tables; NumRelation : INTEGER) : BOOLEAN
 VAR
   i, j : INTEGER;
   RestTable : initially ∅;/*list of join Table */
 BEGIN
   RestTable = ViewDef - Tname;
   Compare RestTable with SALUS
     IF(RestTable[j] = SALUS[i])
           RETURN TRUE; ENDIF;
   RETURN FALSE;
END Compare;
```

```
PROCESS CheckJoin;
  BEGIN
   LOOP
    RECEIVE ViewDef, NumRelation
            FROM Data Warehouse;
        IF( Check(ViewDef, NumRelation) )
        THEN TRIGGER Evaluate;
        ELSE TRIGGER Execute;
    FOREVER;
  END CheckJoin;
```

```
PROCESS Evaluate;
  BEGIN
    LOOP
     FOR(; Compare(ViewDef,NumRelation); );
     ENDFOR;
    FOREVER;
END Evaluate;
```

```
PROCESS Execute;
  BEGIN
    LOOP
     Answer = EXECUTE ΔR;
     REMOVE ΔR FROM SALUS;
     SEND Answer TO Data Warehouse;
    FOREVER;
END Execute;
```

```
MODULE Data Warehouse;
GLOBAL DATA
V:RELATION;/*initialized to the correct view*/
 ViewDef : list of tables
                  /* names of join table */
 NumRelation : INTEGER;
/* number of join Relation to keep the View */
BEGIN  /* start Data Warehouse processes */
  PROCESS(SendViewDef);
  PROCESS(Apply);
END Data Warehouse;
```

```
PROCESS SendViewDef;
  BEGIN
   LOOP
    SEND (ViewDef, NumRelation) TO Source;
   FOREVER;
END SendViewDef;
```

```
PROCESS Apply;
  BEGIN
    RECEIVE Answer FROM Data Source;
    V = V + Answer;
  END Apply;
```

그림 3. 점진적인 뷰 구성 알고리즘

그림 3의 알고리즘을 바탕으로 세 개의 다른 소스 테이블에 네 개의 삽입과 한 개의 삭제가 일어난다고 가정한 예제를 통하여 본 논문에서 제시한 알고리즘을 살펴본다.

r1 : W X r2 : X Y r3 : Y Z
  1 2 2 3 SALUS = ∅

뷰의 정의 $V = \prod_{WY}(r_1 \infty r_2)$라 가정하고 웨어하우스의 실체 뷰(MV)는 MV={(1,3)}으로 초기화되었으며, SALUS는 비어있는 상태로 SALUS= [] 이다. SALUS에 들어가는 갱신 정보는 <갱신 ID, 테이블 이름, 갱신될 값, 연산 종류> 형식으로 가정한다. 갱신 ID는 소스에서 발생한 갱신 정보에 부

여되며, 순차적으로 증가되는 값이다. 테이블 이름은 소스에서 연산이 일어날 테이블 대상이며, 갱신될 값은 삽입과 삭제 연산에 따라서 삽입될 값 혹은 삭제될 값이다. 연산 종류에는 삽입 연산과 삭제 연산이 존재하며, 수정 연산은 삭제와 삽입연산으로 수행한다.

본 논문의 웨어하우스에서는 소스에서 발생한 모든 갱신 정보에 대해서 뷰 정보가 각각 지속적으로 보내지며(worst case), 혹은 소스에서 보내진 뷰 정보 요청 신호를 받은 웨어하우스는 첫 번째 요청 신호에 대해서 단일 사이트이기 때문에 뷰 정보를 단지 한번만 보낸다(best case). 예제에서 소스에서 5개의 갱신들이 U1,U2,U3,U4,U5 순서로 발생되며, 갱신에 대한 뷰 정보 요청신호가 웨어하우스에 도착하는 순서가 U3,U1,U4,U5,U2 이며(도착 순서에 독립적임), 웨어하우스에서 소스로 보내지는 순서도 웨어하우스에 도착한 순서대로 데이터 소스로 보내진다고 가정한다.

소스에서 발생한 각 갱신 정보는 다음과 같다. U1 = insert<r2, (2,4)>, U2 = insert<r1, (3,2)>, U3 = insert<r3, (3,1)>, U4 = insert<r1, (5,2)>와 U5 = delete<r1, (1,2)>이며 SALUS = [<U1, r2, (2,4), insert>, <U2, r1, (3,2), insert>, <U3, r3, (3,1), insert>, <U4, r1, (5,2), insert>, <U5, r1, (1,2), delete>] 이다.

데이터 소스는 뷰 정보를 발생 순서에 무관하게 U3,U1,U4,U5,U2로 받는다고 가정하였다. 나머지 처리과정은 다음 그림 4에 도시되어있다.

소스에서 처리된 결과 응답은 웨어하우스로 보내져서 즉시 실체 뷰에 적용시킬 수 있다. 위의 처리 과정에서 우리는 실체 뷰에 대한 결과가 같은 테이블을 접근하는 갱신들 중에서 순서와 관계없이 처리 가능한 낙관적 방법이 사용될지라도 정확하다는 사실을 알았다.

3. 정확성과 일관성

본 절에서는 웨어하우스에서 뷰가 소스 데이터와 동기화를 이루어 초기화되었다고 가정한다. 정확성은 소스에 존재하는 데이터에 변경이 일어났을 때 뷰가 소스 데이터의 새로운 값을 반영해야함을 의미한다. 특히, 웨어하우스 뷰가 소스 데이터와 일관된 상태를 유지해야한다. 만약, 여러 소스에서 변경이 발생하게되면 웨어하우스의 뷰를 갱신하기 위해서 많은 소스로부터 데이터를 가져와야 하기 때문에 일관성을 유지하기가 어렵다. 이를 위해서 웨어하우스 뷰에대한 일관성을 4단계로 구분하였으며, 각각은 다음과 같이 정의된다[1].

- 집중성(convergence) : 마지막 갱신이후인 모든 행위가 끝난 이후에 뷰가 소스에 있는 데이터와 일관성을 유지한다.
- 약 일관성(weak consistency) : 뷰의 모든 상태는 소스에서 릴레이션들의 유효한 상태에 대응된다. 즉 웨어하우스 각 상태가 어떤 일치된 소스 상태를 반영한다.
- 강 일관성(strong consistency) : 모든 웨어하우스 상태가 소스의 어떤 상태를 반영하며, 웨어하우스의 순서가 소스 상태 순서와 일치되고 모든

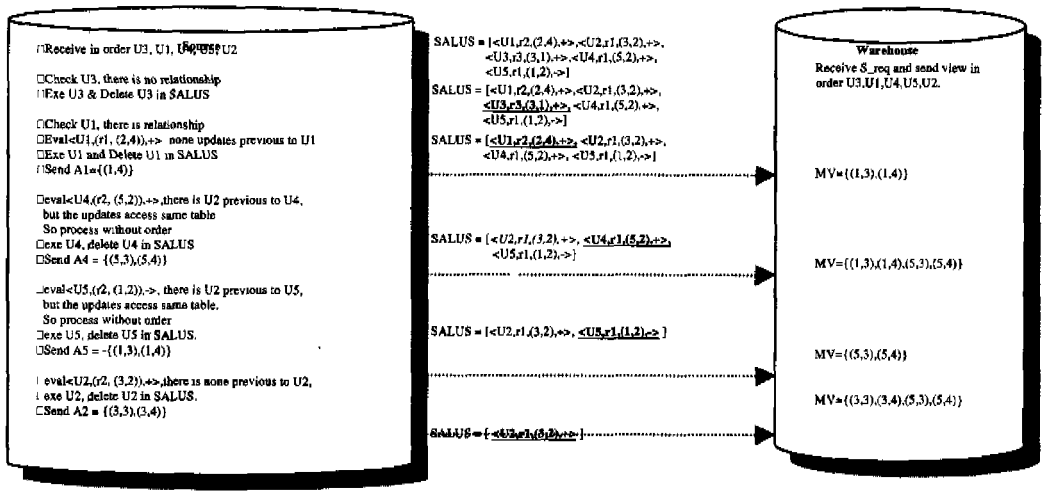

그림 4. SALUS를 이용한 점진적인 뷰 유지 예제

행위가 끝나고 나면 최종 상태가 서로 일치한다.

- 완전성(completeness) : 소스와 뷰 상태 사이에 완전한 순서가 유지되어 매핑이 존재할 경우로서 갱신 하나 일어날 때 마다 뷰 상태를 유지한다.

웨어하우스 실체 뷰의 4단계를 그림으로 살펴보면 그림 5와 같으며, 이전 절의 예제를 통하여 본 논문에서 제시하고 있는 알고리즘은 완전성에 해당되는 일관성을 유지하고 있으며, 기존의 다른 알고리즘에 비해서 제약사항이 감소되었음을 알수 있었다.

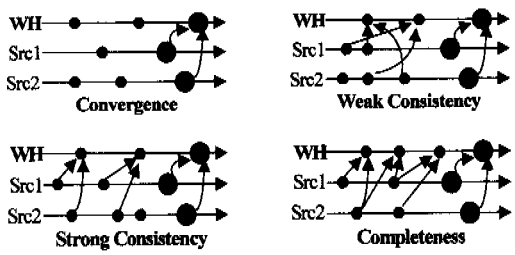

그림 5. 실체뷰의 일관성 단계

Ⅳ. 성능 평가

본 논문의 뷰 유지 방법은 기본적으로 보상 알고리즘을 따르지만, 소스와 웨어하우스의 역할을 변경하여 성능 향상을 도모하였다. 성능 평가의 정당성을 위해서 본 논문에서는 [1]과 같은 평가 기준 환경에서 연구가 이루어졌으며, 또한 뷰 유지 알고리즘의 복잡성이 크게 감소하였다.

전송된 바이트 수에 기반한 성능 평가를 위해서 [1]에서는 뷰와 일련의 갱신 연산들로 구성된 단순 예제를 사용하였다. 본 논문에서도 평가의 공정성을 위해서 [1]과 같은 방법을 이용하여 성능 평가를 하고자한다.

예제1.

소스에 존재하는 기본 테이블 스키마 : r1(W,X), r2(X,Y), r3(Y,Z)

뷰 정의: $V = \prod_{W,Y}(\sigma_{cond}(r_1 \infty r_2 \infty r_3))$

발생된 갱신 정보 : U1 = insert<r1,t1>, U2 = insert<r2,t2>, U3 = insert<r3,t3>

상기 예제에서는 세 개의 갱신 정보를 보였지만 k개의 갱신정보에 대한 평가도 보일 것이다. 다음은 성능 평가를 위해서 수식에 사용될 변수들에 관한 가정이다.

- 각 테이블의 카디널러티는 상수 C로 나타낸다.
- W,Z의 조합 애트리뷰트 크기는 S바이트이다.
- J(ri, a)는 애트리뷰트 a에 대해 특정 값을 갖는 테이블 ri에서 예측 튜플 수, 죠인 요소 J는 모든 경우에 상수로 정의.
- condition cond에서 selectivity는 σ로 나타낸다.($0 \leq \sigma \leq 1$)

RV와 ECA에서는 발생된 하나의 갱신정보에 대해서 소스로부터 웨어하우스로 보내는 메시지 수는 같다. 그러나 보내지는 형태는 "연산종류<테이블 이름, (튜플 값)>"의 형태를 가진다. 이것은 SALUS에서 요청되는 정보의 크기가 RV와 ECA에서 수행된것처럼 데이터 소스에서 웨어하우스로 보내지는 갱신 정보의 크기보다 작은 이유에서이다. 더욱이, 데이터 소스와 웨어하우스 사이에서 보내지는 정보의 크기는 RV와 ECA에서 보다 SALUS에서 항상 작다. 그림 6는 C가 100일 때 갱신 회수와 전송된 바이트 수와의 관계를 보여주고 있다.

위 실험 결과를 통해서 SALUS를 이용한 알고리즘에서는 최악의 경우에도 재 계산 알고리즘에서 발생하는 갱신회수 89개보다 적을 때 좋은 성능을 보였지만 최상의 알고리즘에서는 갱신 정보의 수가 증가함에 따라 오히려 더 좋은 성능의 차이를 보인

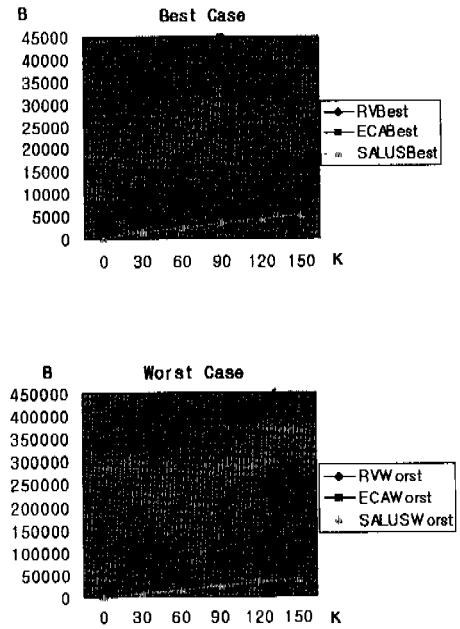

그림 6. 갱신 회수(K)와 전송 바이트 수(B) 관계

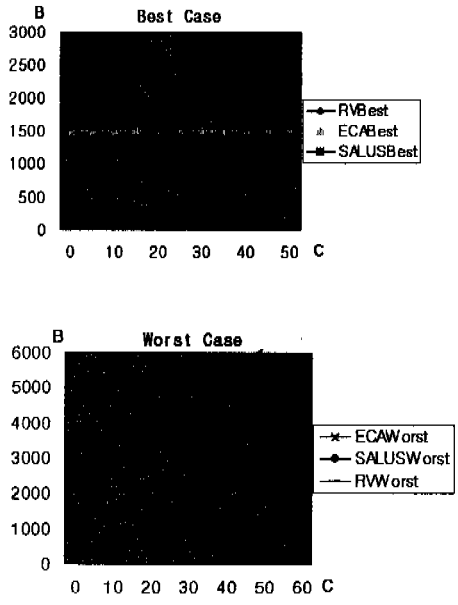

그림 7. 바이트 수(B)와 카디널러티(C)

다는 사실을 알 수 있다. 그림 7에서는 테이블 크기인 카디널러티와 바이트 수의 관계를 소스에서 발생된 일련의 갱신 정보 3개로 가정할 때를 도시하였다. SALUS에서는 상황 여건에 관계없이 모두 좋은 성능을 보이고 있는 것을 실험 결과를 통하여 알 수 있다.

각각의 알고리즘중 재 계산(RV) 알고리즘은 뷰 갱신의 빈도수에 따라 영향을 받으며, 보상 알고리즘(ECA)에서는 갱신 정보 도착 시간에 의존적이다. SALUS에서는 갱신된 정보의 접근 테이블에 따라 성능의 차이를 보이게된다. 입·출력 발생에 따른 성능 평가 과정은 ECA의 처리 과정과 같게되며, 결과도 유사하게 되므로 본 논문에서는 다루지 않았다. 과정 및 결과에 관한 자세한 사항은 [1, 6]에 언급되었다.

그림 8에서는 본 논문에서 제시하고있는 알고리즘의 정확성 및 일관성을 다른 알고리즘과 비교한 상태도를 나타내었다. 이전 절의 예제를 통하여 본 논문에서 제시하고 있는 알고리즘은 완전성에 해당되는 일관성을 유지하고 있으며, 기존의 다른 알고리즘에 비해서 제약사항이 감소되었음을 알수 있었다.

V. 결론 및 향후 연구

데이터 웨어하우스는 의사 결정을 지원하기 위한

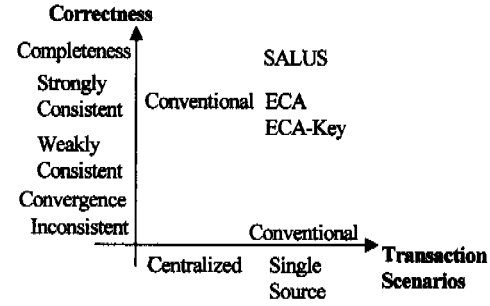

그림 8. 정확성에 따른 알고리즘 비교

주제지향적이며, 통합되며, 비휘발성이며, 시간 개념을 지닌 데이터 집합이다. 더욱이 웨어하우스는 사용자에게 효율적으로 결과를 응답해야하며, 질의 처리를 최적화하기 위해서 데이터 웨어하우스는 실체 뷰들을 가지고 있다. 그러므로 이를 위해 데이터 웨어하우스는 데이터 소스와 데이터 웨어하우스 뷰 사이에 유용한 상태를 유지할 필요가 있다.

본 논문에서는 실체 뷰를 유지하는 방법가운데 단일 소스 환경에서 뷰 유지를 보다 효율적으로 수행할 수 있는 새로운 알고리즘을 제시하였다. 또한, 우리는 RV와 ECA로 알려진 알고리즘과 본 논문에서 SALUS를 이용하여 제시하고 있는 알고리즘의 성능을 비교 분석하였으며, 정확성에 대한 상태도 나타내었다. 향후 연구로서는 중앙 집중화된 환경의 실체 뷰 유지에서 분산된 환경의 데이터 웨어하우스 실체 뷰 유지로 확장과 관리가 용이한 알고리즘에 대한 연구가 필요하다.

참 고 문 헌

[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," *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 405-416. 1997.

[3] L. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey, "Algorithms for deferred view maintenance," *Proceedings of ACM SIGMOD 1996 International Conference*, Montreal,

Canada, June 1996.

[4] T. Griffin and L. Libkin. "Incremental Maintenance of Views with Duplicates," *In Proc. of ACM SIGMOD* 1995.

[5] 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.

[6] D. Agrawal, A. El Abbadi, A. Singh, T. Yurek. "Efficient View Maintenance at Data Warehouses,". *Proceedings of ACM SIGMOD* 1997.

[7] 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.

[8] D. Agrawal, A. El Abbadi, A. Singh, and T. Yurek, "Efficient View Maintenance at Data Warehouse", *In proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 417-427, 1997

[9] Rob Mattison, Web Warehousing and Knowledge Management, *McGraw-Hill*, 1999.

이 현 창(Hyun-Chang Lee) 준회원

1993년 2월 : 원광대학교 컴퓨터 공학과 학사

1996년 2월 : 홍익대학교 전자계산학과 석사

1999년 3월~현재 : 홍익대학교 전자계산학과 박사 과정

<주관심 분야> 웹 웨어하우징, 데이터 웨어하우스, 데이터 마이닝, 전자상거래, 멀티미디어, 주기억 데이터베이스,

김 충 석(Chung-Seok Kim) 정회원

1986년 2월 : 홍익대학교 전자계산학과 졸업(이학사)

1988년 2월 : 홍익대학교 전자계산학과 졸업 (이학석사)

1993년 2월 : 홍익대학교 전자계산학과 졸업 (이학박사)

1990년~현재 : 신라대학교(구. 부산여자 대학교) 컴퓨터 정보공학부 부교수

<주관심 분야> Distributed Object Processing, Intranet Solution, OOP

김 경 창(Kyung-Chang Kim) 정회원

현재 홍익대학교 컴퓨터 공학과 부교수

한국통신학회지 논문지 제 21권 7호 참조