부가파일을 이용한 실체 뷰 관리 기법의 성능평가

정웅교[1)0] 김진호[1)] 이우기[2)]
[1)]강원대학교 컴퓨터과학과,
[2)]성결대학교 컴퓨터학부
unge91@cs.kangwon.ac.kr, jhkim@kangwon.ac.kr, wook@sungkyul.edu

Performance Evaluation on Materialized View Management using Differential Files

Woong-Kyo Chung[1)0] Jin-Ho Kim[1)] Woo-Key Lee[2)]
[1)]Dept. of Computer Science, Kangwon National University
[2)]Dept. of Computer Science, Sungkyul University

요 약

데이터웨어하우스내에서 실체뷰는 소스 데이터에서 갱신이 발생하였을 경우 질의에 정확한 결과를 제공하기 위하여 릴레이션의 변경사항과 동일하게 갱신되어야 한다. 이 뷰를 갱신할 때 전체 릴레이션이 아닌 갱신된 부분만 이용하여 점진적으로 뷰를 관리하는 것이 효과적이다. 뷰의 점진적인 갱신 방법중에서 부가화일을 이용한 방법은 부가적으로 유지하는 정보의 양이 적고 뷰 관리 비용이 적게 든다는 장점이 있다. 이 논문에서는 이 방법에 의해 뷰를 관리할 때의 성능을 평가하기 위하여 이 방법의 비용 모델을 분석적으로 제시하였으며, 이 모델을 기반으로 성능을 비교.분석하여 부가 파일을 이용한 방법이 다른 기존의 방법보다 성능이 우수함을 보였다.

1. 서 론[1)]

데이터 웨어하우스란 의사결정에 필요한 정보를 제공하기 위해 여러 사이트에 분산된 데이터베이스들로부터 데이터를 수집하고 분석한 데이터를 모아 놓은 거대한 데이터의 저장소이다 [1][2]. 데이터 웨어하우스 내의 정보는 사용자 질의에 대한 빠른 응답을 위해 실체뷰(materialized view)로 저장된다. 실체뷰는 자주 요구되는 질의에 관계되는 투플들이 요약된 형태로 실제로 저장된 테이블이다. 실체뷰는 기본 릴레이션으로부터 추출된 요약된 결과를 별도로 저장하고 있으므로 소스 데이터에서 삽입, 삭제, 수정이 발생하였을 경우, 질의에 정확한 결과를 제공하기 위하여 기본 릴레이션의 변경사항과 동일하게 갱신되어야 한다.

소스 데이터의 갱신에 따라 뷰를 최신의 정보로 유지하는 것을 뷰 관리(view maintenance)라 한다. 뷰 관리 기법으로는 뷰 재정의 기법(view redefinition)과 점진적 뷰 관리 기법(incremental view maintenance)이 연구되어 왔다[3][4]. 뷰 재정의 기법은 데이터 소스의 변경사항 발생시 뷰의 정의에 따라 기본 릴레이션으로부터 뷰를 재계산하는 방법이다[3]. 뷰 재정의 기법은 기본 릴레이션의 접근으로 인해 많은 계산 비용이 소요된다는 단점이 있다.

이에 비해 점진적 뷰 관리 기법은 기본 릴레이션의 변경 사항만을 가지는 보조 릴레이션(auxiliary relation) 또는 부가 파일(Differential File)을 이용하여 데이터 소스의 접근 없이도 뷰에 변경 사항을 효율적으로 반영하는 뷰의 자체적 관리 기법이다 [1][6]. 이 중에서 부가 파일을 이용한 뷰 관리 기법은 추가로 유지하는 정보의 양이 적고 그 수행 시간이 적게 걸리는 효과적인 방법으로 제시되었다. 이 논문에서는 뷰의 점진적 관리 기법들의 성능을 평가하기 위한 비용 모델을 분석적으로 제시하였다 (지면 관계상 부가화일을 위한 조인 뷰 관리 방법에 대한 비용 모델만 소개한다). 이 모델을 기반으로 뷰 관리 기법들의 성능을 비교.평가하였다. 성능 평가 결과 부가화일을 이용한 방법이 기존의 다른 뷰 관리 방법보다 성능이 더 우수함을 보였다.

1) 이 논문은 첨단정보기술연구센터(AITrc)를 통하여 한국과학재단의 지원을 받았음.

2. 관련연구

데이터 웨어하우스에서 기본 릴레이션의 변경에 따라 실체뷰를 최신의 정보로 유지하는 뷰 관리에 관한 연구는 폭 넓게 연구되고 있다 [1][5]. 그 중에서도, 뷰 관리 비용을 줄이기 위하여 기본 릴레이션의 변경사항만을 뷰에 점진적으로 반영하는 뷰의 자체적 관리에 관한 연구가 관심 깊게 연구되고 있다. 뷰의 자체적 관리 방법은 보조 릴레이션을 사용하는 방법과 부가 파일을 사용하는 방법으로 크게 구분된다.

이중에서 부가 파일을 이용하는 방법은 각각의 기본 테이블에 대한 변경 사항들만 별도의 부가 파일에 유지한 후 일정 시간마다 이 변경 사항들을 뷰에 반영하는 방법이다[6]. 부가 릴레이션에는 기본 테이블의 변경 사항만 저장하므로 유지해야 할 정보의 양이 더 작다. 또한 기본 테이블이나 중간 연산 결과를 접근하지 않고, 부가 릴레이션만으로 최종 뷰를 수정하므로 처리 시간이 적게 소요된다는 장점이 있다. 부가 파일을 이용한 뷰 관리 기법에 대한 기존 연구로 [6]에서는 기본 릴레이션의 변경 사항을 유지하는 부가 파일과 다른 기본 릴레이션에 새로운 투플이 삽입될 때 참조 무결성 제약 조건을 이용하여 이 투플과 조인될 투플을 찾아 별도의 조인 부가 파일에 유지하여, 기본 릴레이션들을 전혀 접근하지 않고 조인 뷰를 관리하는 방법을 제안하였다. 이 방법은 기본 릴레이션보다는 크기가 매우 작은 부가 파일과 조인 부가 파일만 사용하므로 뷰 갱신 시간을 현저히 줄일 수 있다는 장점이 있다.

이러한 뷰 관리 기법들은 효과적으로 뷰를 갱신할 수 있지만 그 관리 비용에 대해서는 자세하게 연구되지 않았다. 따라서 이 연구에서는 뷰 관리 기법들의 성능을 평가하고 비교.분석하기 위하여 그 비용 모델을 분석적으로 제시하고자 한다. 또 이 비용을 이용하여 뷰 관리기법들의 성능을 평가하고자 한다.

3. 부가 파일을 이용한 자체적 복합 뷰 관리 기법

3.1 실체 뷰와 부가 파일

이 논문에서 기본 릴레이션은 Ri(TIDi, Ai, AFK)와 Rj(TIDj, Aj)로 이뤄지며, TID는 기본 릴레이션들의 투플 식별하기 위한

기본 키라고 정의한다. Ai와 Aj는 각 릴레이션의 속성들의 집합이며, AFK는 Ri의 외래 키로써 Rj의 TIDj와 연계된다. 데이터 웨어하우스에서의 뷰는 V(VIDv, Av)로서 정의되며 저장된 실체 뷰이다. 이 때, VIDv는 뷰 식별자이며, Av는 뷰의 속성들의 집합으로 Ai와 Aj의 부분집합으로 구성된다.

기본 릴레이션 Ri(TIDi, Ai)의 부가파일은 dRi(TIDi, Ai, operation- type, TSd)으로 정의된다. 부가 파일의 구성 요소 중 Operation-type은 투플 변경에 적용된 연산을 나타내며 insert와 delete 중의 하나의 타입을 가진다. 'update'의 경우 'delete'와 'insert'의 조합으로 간주한다. TSd는 기본 릴레이션의 투플에 변경연산이 적용된 시간(time-stamp)을 나타낸다.

3.2 Join 뷰 관리

데이터 웨어하우스에서의 뷰는 star schema나 snowflake schema의 외래 키에 의한 join뷰로 간주될 수 있다. 이 외래키에 관한 참조 무결성 규칙은 그 외래키에 부합하는 기본 키값이 대응하는 릴레이션에 존재해야 함을 나타낸다. 이는 외래키와 조인될 투플과의 관계로 해석할 수 있다. 따라서 이 참조 무결성 규칙을 활용한다면 join 뷰의 관리를 간단하게 할 수 있다.

이 같은 이유로 본 논문에서는 기본 릴레이션간에 참조 무결성 규칙이 존재한다고 가정한다. 만일 기본 릴레이션 Ri와 Rj에 대하여 AFK ⊆ TIDj와 같은 참조 무결성 규칙이 존재할 경우, Ri 는 참조하는 릴레이션이고 Rj는 참조되는 릴레이션에 해당된다.

기본 릴레이션 Rj의 조인 부가 파일은 jRj(TIDj, Aj, TSj)로 정의한다. jRj에 투플을 삽입할 때 dRj에 중복된 투플이 존재하는지 여부를 점검한다. jRj와 dRj는 같은 릴레이션 Rj로부터 유도되므로 같은 내용의 투플을 가질 수 있다. 참조되는 릴레이션의 조인 부가 파일과 부가 파일에 동일한 투플이 존재한다면, 조인 뷰의 갱신시 조인 부가 파일과 참조하는 릴레이션의 부가 파일과의 조인 연산으로 얻어진 결과와 참조되는 릴레이션의 부가 파일내의 변경 투플을 뷰에 반영하는 절차에 있어 중복된 과정이 발생하게 된다. 따라서 jRj내에서 dRj에 존재하는 투플로 인한 중복된 투플을 제거하는 과정이 필요하다.

그림 1은 기본 릴레이션의 변경에 따라 join 뷰를 갱신하는 과정을 보여주고 있다. 그림 1에서 보는 바와 같이, 뷰의 갱신은 기본 릴레이션을 접근하지 않고 부가 파일과 조인 부가 파일을 통해서만 수행되므로 뷰가 자체적으로 관리된다.

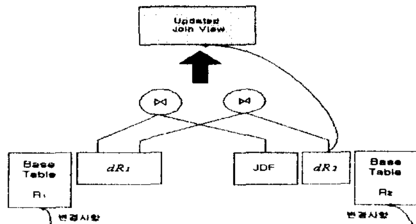

그림 1 Join 뷰의 갱신

Join 뷰에 대한 갱신은 다음과 같이 수행된다. Temp1은 부가 파일과 조인 부가 파일의 조인과 union 연산을 수행한 중간 결과이며, p는 참조 무결성 규칙 조건을 나타내며, VA는 뷰의 속성들을 표현한다. 그리고 ⋈는 조인연산을 나타내고, ∏는 프로젝션 연산을 표현한다.

1. dR1의 투플을 R2의 사이트로 전송
2. Temp1 ←(∏VA (dR1⋈p dR2))∪(∏VA(dR1⋈p JDF))
3. Temp1과 dR2의 투플을 뷰의 사이트로 전송
4. 뷰 갱신

4. 뷰 관리 기법의 비용 모델

이 장에서는 뷰의 관리에 관한 비용모델을 설명한다(지면 관계상 조인 뷰의 비용 모델만 소개한다). 비용모델은 뷰 관리의 갱신 절차에 관여되는 처리비용 및 전송비용으로 구성된다.

4.1 Parameter

비용모델에 사용되는 매개변수와 식은 다음과 같다.

Rr : Rr의 기본 릴레이션 (r=i,j)
dRr : 기본 릴레이션의 부가 화일 dRr (r=i,j)
jRr : 참조되는 릴레이션의 조인 부가 화일
N(T) : 테이블 T의 투플의 수 (R, DF, JDF T)
W(T) : 테이블 T의 투플의 크기 (R, DF, JDF T)
P(T) : 해당 투플이 T에 존재할 확률
B : 블록 크기(bytes)
$C_{I/O}$: I/O비용 (ms/block)
C_{COM} : 전송율 (bits/s)
p_r : projection으로 축소될 투플의 사이즈의 비율
a_r : 그룹핑으로 축소될 투플의 비율
s_r : 두 릴레이션간에 매칭되는 투플의 비율
d_f : 부가 화일내의 중복 투플 비율
key_size : 릴레이션의 키에 대한 사이즈
bsize(t) : t의 블록크기 계산
Read(t) : 테이블 t의 투플을 읽음
Sort(t) : 테이블 t의 투플을 정렬
Join(t, t1) : 조인하기 위해 조인대상인 테이블 t와 t1을 읽음
Join_write(t, t1) : 테이블 t와 t1의 조인 결과를 기록
Agg(t) : t의 집계값을 계산
Update(t) : t의 변경사항을 갱신
Comm(t) : t의 전송비용 계산

아래의 식 [식 4.1.1], [식 4.1.2], [식 4.1.3]은 다음 절에서 사용될 각각 의 릴레이션의 변경사항에 대한 블록 크기를 계산하는 bsize(t)를 표현하고 있다.

bsize(dRi) = N(dRi)*W(dRi)/B [식4.1.1]
bsize(dRj) = N(dRj)*W(dRj)/B [식4.1.2]
bsize(jRj) = N(jRj)*W(jRj)/B [식4.1.3]

4.2 Join 뷰

Join 뷰에서의 비용함수는 dRi, dRj, jRj을 소팅하는 비용과, dRi의 투플을 참조되는 릴레이션으로 전송하는 비용, 조인결과를 디스크에 저장하고, 뷰가 존재하는 사이트로 보내는 전송비용, 그리고 뷰를 갱신하는 비용으로 구분된다. 전송비용과 관련하여 각각의 릴레이션은 3개의 사이트에 존재한다고 가정한다. 따라서 dRi의 내용을 Rj의 사이트로 전송하고, Rj의 사이트에서의 조인결과를 뷰로 전송하는 비용이 필요하게 된다.

■ 소팅비용 = Sort(dRi) + Sort(dRj) + Sort(jRj)
Sort(dRi) = 2*d_f*bsize(dRi)*log_k(d_f*bsize(dRi)) [식4.2.1]
Sort(dRj) = 2*d_f*bsize(dRj)*log_k(d_f*bsize(dRj)) [식4.2.2]
Sort(jRj) = 2*bsize(jRj)*log_k(bsize(jRj)) [식4.2.3]

■ 조인비용 = (dRi,dRj)간의 조인비용+(dRi,jRj)간의 조인비용
= Join(dRi,dRj) + Join_write(dRi,dRj)
+ Join(dRi,jRj) + Join_write(dRi,jRj)

Join(dRi,dRj) = d_f*bsize(dRi)+ d_f*bsize(dRj) [식4.2.4]
Join_write(dRi,dRj) = (d_f*N(dRi_ins) * N(dRj_ins)
/(N(Rj)+ N(dRj_ins)))*(W(dRi)+ W(dRj))*p_r/B [식4.2.5]
※ bsize(dRi_dRj_join_write) = Join_write(dRi,dRj)
Join(dRi,jRj) = d_f*bsize(dRi)+ bsize(jRj) [식4.2.6]
Join_write(dRi,jRj)=d_f*(N(dRi)-N(dRi_ins)*N(dRj_ins)
/(N(Rj)+ d_f*N(dRj_ins)))*(W(dRi)+ W(jRj))*p_r/B [식4.2.7]
※ bsize(dRi_jRj_join_write) = Join_write(dRi,jRj)

■ 전송비용 = dRi 전송비용 + 조인결과 전송비용

$Comm(dRi)=2*d_f*bsize(dRi)+d_f*bsize(dRi)*B/C_{COM}$ [식4.2.8]

$Comm(join_write(dRi,dRj))=2*bsize(dRi_dRj_join_write) + bsize(dRi_dRj_join_write)*B/C_{COM}$ [식4.2.9]

$Comm(join_write(dRi,jRj))=2*bsize(dRi_jRj_join_write) + bsize(dRi_jRj_join_write)*B/C_{COM}$ [식4.2.10]

■ 뷰 갱신비용 = Update(V)

$Update(V) = 2*bsize(dRi_dRj_join_write)+2*d_f*(N(dRi_ins) - N(dRi_ins)*N(dRj_ins)/(N(Rj)+N(dRj_ins))) * (W(dRi) + W(jRj))*p_r/B + d_f*N(dRi_del+Rj_del) * (\log_{B*0.75/k_size}(N(V)*key_size)+2)$ [식4.2.11]

■ 전체비용 = 소팅비용 + 조인비용 + 전송비용 + 뷰갱신비용
즉, [식4.2.1]에서[식4.2.11]의 값을 모두 더한 값이다.

5. 뷰 관리 기법의 성능 평가

실험은 릴레이션 크기와 부가 파일의 투플 수를 증가시키면서 각 뷰 관리 방법에 소요되는 비용을 계산한다. 변경 투플수 즉, 부가 파일의 insert와 delete의 투플 수를 100에서 1,000,000까지 증가시키면서 소요비용을 측정하며, 기본 릴레이션의 크기를 100에서 1,000,000까지 증가시키면서 뷰 관리 비용이 어떻게 변화하는지 측정한다. 성능평가의 비교를 위해 보조 릴레이션을 이용한 뷰 관리방법과 본 논문에서 제안한 방법을 비교한다.

5.1 변경 투플수 변화에 따른 성능 분석

그림 2는 변경 투플수의 변화에 따른 join 뷰에 대한 성능평가 결과를 보여주고 있다. 변경 투플의 수가 적을 시에는 보조 릴레이션을 사용한 방법과 큰 차이가 없으나 변경 투플의 수가 증가하면서 블록 I/O의 수의 차이가 커짐을 확인할 수 있다.

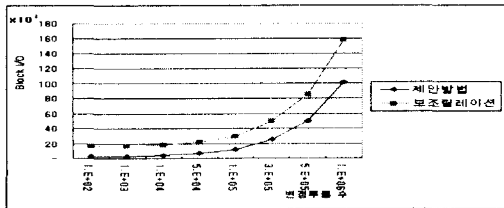

그림 2 join 뷰 성능비교 (변경투플)

5.2 릴레이션 크기의 변화에 따른 성능분석

릴레이션 크기의 변화를 통한 성능분석에서는 변경 투플 수를 100,000개로 고정시킨 후 각각의 복합 뷰 연산에 관여하는 릴레이션 크기를 동일하게 증가시키면서 성능을 비교한다. 이 때 릴레이션에 존재하는 투플의 수는 100에서 1,000,000개까지 증가시킨다.

그림 3은 릴레이션의 크기 증가에 따른 블록 I/O 수를 보여주며, 릴레이션의 크기의 변화에 따라 각 연산의 블록 I/O 수가 증가한다.

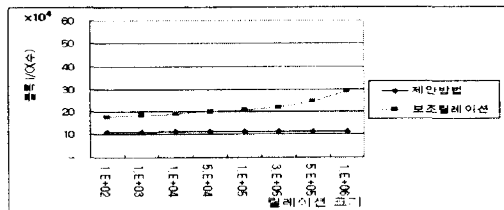

그림 3 join 뷰 성능비교 (기본릴레이션 크기)

이것은 릴레이션의 증가에 따라 뷰에 존재하는 투플의 수가 증가하기 때문이다. 그러나 5.1절의 변경 투플 수의 증가에 따른 변화보다 변화 폭이 적음을 보인다. 그 이유는 제안된 방법과 점진적 뷰 관리 방법이 뷰 재정의 방법과 같이 릴레이션의 크기에 영향을 많이 받기보다 변경 투플의 수에 영향을 받기 때문이다. 즉 릴레이션에 접근하지 않고 변경사항만을 가지고서 뷰를 갱신하므로 릴레이션 접근비용이 들지 않기 때문이다. 보조 릴레이션을 이용한 방법과 제안된 방법과의 비교에 있어 보조 릴레이션을 이용한 방법은 릴레이션의 크기가 커질수록 저장되는 중간연산 결과가 증가하므로 제안된 방법보다 계산비용이 더 증가하게 된다.

6. 결론

데이터 웨어하우스 환경에서 실체뷰에 대한 데이터 소스들은 여러 기본 릴레이션에 분산, 저장되기 때문에 기본 릴레이션에 대한 변경 연산을 처리하기 위하여 많은 어려움이 존재한다. 따라서 뷰를 효율적으로 관리하기 위해 기본 릴레이션의 접근 없이 뷰를 자체적으로 관리할 수 있는 방안이 필요하다.

본 논문에서는 뷰 관리에 효과적이라고 알려진 점진적인 뷰 관리 기법중에서 부가 파일을 이용하는 방법의 성능을 평가하였다. 이 방법은 기본 릴레이션을 전혀 접근하지 않고 부가 파일만을 이용하여 뷰를 관리함으로써 기본 릴레이션에 대한 일반 업무 처리를 지연시키지 않아 처리 성능이 우수하며 기본 릴레이션에 대한 가용성을 높일 수 있는 효과적인 방법이다. 이 방법의 성능을 평가하기 위해 각 뷰 관리 연산에 대한 비용 모델을 분석적으로 설계하였으며 이에 대한 성능을 평가하였다. 또한 점진적 뷰 관리 기법으로 널리 사용되는 보조 릴레이션을 이용하는 방법과 성능을 비교.분석하였다. 성능평가 결과 부가 파일을 이용한 뷰 관리 기법이 보조 릴레이션을 사용하는 경우보다 성능이 더 우수함을 보였다.

7. 참고 문헌

[1] A. Gupta and I. S. Mumick, "Maintenance of Materialized Views : Problems, Techniques, and Applications," In Proc. IEEE DEBU, Vol. 18, No. 2, pp. 3-18, 1995.

[2] M. Mohania, S. Samtani, J. Roddick and Y. Kambayashi, "Advances and Reaserch Directions in Data Warehousing Technology," AJIS, Vol. 7, No. 1, pp. 41-59, 1999.

[3] L. Colby, A. Kawaguchi, D. Lieuwen, I. S. Mumick and L. Ross, "Supporting Mutiple View Maintenance Polices," In Proc. of ACM SIGMOD Conf., pp. 405-416, 1997.

[4] A. Gupta, H. Jagadish and I. S. Mumick, "Data Integration Using Self-maintainable Views," In Proc. EDBT, pp. 140-144, 1996.

[5] T. Griffin and L. Libkin, "Incremental Maintenance of Views with Duplicates," In Proc. of ACM SIGMOD Conf., pp. 328-339, 1995.

[6] W. Lee, "On the Independence of Data Warehouse from Databases in Maintaining Join Views," In Proc. Lecture Notes in Computer Science, pp. 86-95, 1999.

[7] C.Y. Chan and Y.E. Ioannidis, "Bitmap Index Design and Evaluation," In Proc. Int'l. Conf. on Management of Data, ACM SIGMOD, pp. 355-366, Seattle, Washington, USA, June 1998.