STAREX 교환기 데이터베이스 접근의 성능 향상 기법

이 규 영, 권 경 인, 조 시 철
LG정보통신(주) 교환연구소

A Technique to Improve the Performance of Database Access in STAREX Switching Systems

Kew-Yung Lee, Kyoung-In Kwon, Si-Cheol Cho
Switching System Research Lab., LGIC. Ltd.

요 약

STAREX 교환기의 DBMS는 교환기 시스템의 호처리, 운용, 보전 등에 관련된 모든 데이터를 유지하면서, 응용 프로그램 데이터의 효율적 지원, 데이터의 일관성 유지, 데이터의 백업, 데이터의 무결성 진단 및 복구 등의 종합적인 관리를 수행하는 시스템 소프트웨어이다. 또한, DBMS는 교환기의 실시간 처리 요구를 만족시키기 위하여 여러 가지 기능들을 제공한다. 그러나, 이러한 기능들은 응용 프로그램들이 얼마나 효율적으로 사용하느냐에 따라 성능이 크게 좌우된다. 본 논문에서는 STAREX 교환기의 DBMS가 제공하는 실시간 처리 기능들을 소개하고, 교환기의 성능을 향상시키기 위하여 응용 프로그램들이 효율적으로 데이터베이스에 접근하는 방안을 제시한다.

I. 서론

STAREX 교환기의 DBMS(Database Management System)는 교환기 시스템의 호처리, 운용, 보전 등에 관련된 모든 데이터를 유지하면서, 응용 프로그램 데이터의 효율적 지원, 데이터의 일관성 유지, 데이터의 백업, 데이터의 무결성 진단 및 복구 등의 종합적인 관리를 수행하는 시스템 소프트웨어이다.

이러한 교환기 DBMS의 가장 중요한 요구 사항은 교환기 시스템에서 동시 다발적으로 발생하는 데이터 접근 요구의 실시간 처리이다. 예를 들면, 호처리 과정에서 필요한 데이터 접근 요구는 실시간 응답이 필수적이다. 따라서, 교환기 DBMS는 실시간 처리를 하기 위하여 여러 가지 기능들을 제공한다. 그러나, 많은 교환기 응용프로그래머들은 DBMS가 제공하는 기능들을 효율적으로 사용하지 않은 경우가 많아 교환기 DBMS의 접근 성능 향상 기회를 잃고 있는 경우가 허다하다. 따라서, 응용 프로그램들이 교환기 DBMS가 제공하는 기능들을 얼마나 잘 이해하고 이용하느냐에 따라 교환기의 성능이 크게 좌우한다고 할 수 있다.

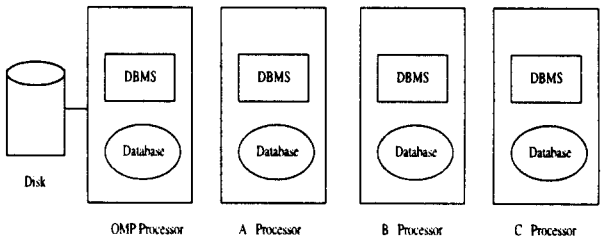
본 논문에서는 데이터베이스 접근 성능을 향상시키기 위하여 응용 프로그램들이 효율적으로 데이터베이스에 접근하는 방안을 제시한다. 제2장에서는 실시간 처리를 하기 위한 DBMS의 구조를 살펴보고, 제3장에서는 DBMS가 실시간 처리를 하기 위하여 제공하는 기능들을 소개하고 이를 효율적으로 사용하는 방안을 제시하며, 마지막으로 제4장에서는 결론을 맺는다.

II. 실시간 처리 지원을 위한 DBMS의 구조

본 절에서는 STAREX 교환기의 DBMS의 구조와 실시간 처리를 하기 위한 특성을 설명한다.

그림 1은 STAREX 교환기 시스템 내의 DBMS의 구조를 보여주고 있다. 교환기 시스템은 여러 프로세서들로 구성되어 있는 분산 구조로 되어 있으며, 이 중, OMP(Operation and Maintenance Processor)는 시스템의 운용, 유지 및 관리를 하는 프로세서로서, 디스크를 구비하고 있다. DBMS와 각 프로세서에서 필요로 하는 데이터베이스는 실시간 처리를 하기 위하여 각 프로세서의 주기억 장치에 상주한다. 또한, 각 프로세서의 데이터에 대한 백업 및 로딩을 하기 위하여 각 프로세서의 데이터베이스는 OMP의 디스크에도 존재한다.

<그림 1> STAREX 교환기의 DBMS 구조



교환기 DBMS는 사용자 인터페이스 처리 부분과 질의 처리 부분, 그리고, 데이터베이스 엔진 부분으로 구성된다. 사용자 인터페이스 처리 부분은 오프 라인 상태에서 작성된 사용자의 질의어를 분석하고 필요한 정보를 구성하여 질의 처리기에게 질의 처리를 요구하는 기능을 담당한다. 질의 처리 부분은 데이터 접근 명령에 대한 분석 및 처리를 담당한다. 데이터베이스 엔진 부분은 접근 관리기, 동시성 제어기, 데이터 관리기, 원격 데이터 관리기와 디스크 데이터에 대한 백업 처리를 위해 로그를 생성하고 OMP로 생성된 로그를 전송하기 위한 로그 관리기로 구성된다. 데이터베이스 엔진 부분은 각 프로세서마다 실장되며, OMP에는 디스크 접근을 담당하여 데이터에 대한 백업 기능과 시스템 재시동 시 데이터 회복 기능을 위한 백업 관리기가 추가로 실장된다.

분산 시스템 환경하에서 각 프로세서에 중복되어 존재하는 데이터를 관리하기 위한 중복 데이터 관리기는 OMP에만 추가로 실장 되는 데, 이것은 중복 데이터를 관리하는 데 있어 중복된 프로세서들의 데이터 일치성 보장을 위한 정보 저장과 중복 데이터 변경에 대한 백업을 위한 디스크가 OMP에만 구비되어 있기 때문이다.

이와 같은 구조를 갖는 DBMS는 실시간 처리를 하기 위해서 다음과 같은 특성들을 갖는다.

우선, 데이터베이스를 주기억장치에 상주시킨다. 주기억장치

에 대한 접근의 속도는 디스크 접근에 비하여 속도가 매우 빠르다[1]. 만약, 디스크 데이터베이스 관리 시스템의 방식을 사용한다면 디스크의 검색 및 입출력 시간 등의 오버헤드 때문에 동시 다발적으로 발생하는 데이터 오퍼레이션을 실시간으로 처리하는 것은 어렵다.

둘째, 데이터베이스가 분산 구조로 되어 있다. 교환기 데이터베이스는 분산 구조를 갖는 교환기 시스템을 지원하기 위해서 각 프로세서에 관련된 데이터를 프로세서에 따라 분산시키는 것이 바람직하며 이러한 데이터를 조작하여 실시간 서비스를 제공하기 위하여 교환기 DBMS 역시 분산 구조로 구현되어 있다.

셋째, 데이터 접근 시 튜플 단위로 이루어진다. 교환기 DBMS에서 사용하는 데이터 모델은 현재까지 가장 널리 사용되고 있는 관계형 모델(relational model)[2,3]이다. 이러한 모델을 사용하고 있는 교환기는 응용 프로그램의 접근 결과를 빠른 속도로 응답하기 위하여 접근 결과를 튜플 단위로 이루어지게 하고 있다.

III. 실시간 지원을 위한 DBMS 기능과 효율적 사용 기법

STAREX 교환기에서는 각종 가입자 정보 등을 관리하기 위해 수백 개의 릴레이션으로 이루어진 복잡한 데이터베이스가 사용된다. 또한, 이 릴레이션들은 복수 개의 분산 프로세서들로 이루어진 교환기의 분산 구조 때문에 적절한 프로세서에 분산되어야 한다. STAREX 교환기의 DBMS는 이러한 데이터베이스의 실시간 처리를 위하여 여러 가지 기능들을 제공한다. 본 장에서는 실시간 처리 기능들을 소개하고 이러한 기능들을 효율적으로 사용하는 기법을 제시한다.

3.1 릴레이션의 저장 위치

STAREX 교환기의 데이터베이스를 효과적으로 유지하고 관리하는 것은 쉬운 일이 아니며 데이터를 어떻게 분산시킬 것인가 하는 문제는 상당히 중요하다. 이것은 DBMS의 접근 성능을 크게 좌우할 수 있는 요소이므로 데이터베이스의 설계와 분산 과정에서 여러 가지 요소들을 충분히 고려해야 한다.

교환기 데이터베이스에 포함되는 릴레이션은 한 개 또는 다수의 프로세서의 메모리에 저장되어질 수 있는 데, 교환기 데이터베이스의 릴레이션들은 릴레이션이 저장되는 프로세서의 위치에 따라 다음과 같은 종류들로 구분된다.

1) NOR(No Replicated)
   하나의 릴레이션이 한 프로세서에만 존재하는 가장 기본적인 형태의 릴레이션
2) REP(Replicated)
   하나의 릴레이션이 여러 프로세서에 동일하게 저장되는 형태의 릴레이션
3) LOR(Local Replicated)
   동일한 구조, 즉 스키마와 최대 튜플 수가 같은 동일한 이름의 릴레이션이 다수의 프로세서들에 존재하지만, 각 프로세서에 저장되는 튜플들은 서로 다른 값을 갖는 릴레이션
4) DIS(Distributed)
   하나의 릴레이션을 튜플의 교집합이 없이 수평적으로 분할하여 각각을 여러 프로세서들에 분산시킨 형태의 릴레이션

사용자들은 릴레이션 등록 시, 위의 종류 중 하나를 선택하여 등록하여야 한다. 또한, 그 릴레이션이 존재할 프로세서에 따라 지역 프로세서(local processor)와 광역 프로세서(global)를 등록하여야 한다. 지역 프로세서란 해당 릴레이션이 존재할 프로세서를 말하며, 광역 프로세서란 해당 릴레이션이 존재하지는 않지만 그 릴레이션을 읽고 쓸 수 있는 프로세서를 말한다. 따라서, 사용자들은 등록할 릴레이션의 특성을 잘 고려하여 릴레이션의 종류를 선택하고, 해당 릴레이션에 대한 지역 프로세서들과 광역 프로세서들을 결정하여야 한다.

지역 프로세서들에서 수행되는 응용 프로그램들은 자기 자신의 프로세서에서 해당 릴레이션을 접근하면 되므로, 릴레이션에 대한 접근이 빠르게 이루어진다. 그러나, 광역 프로세서에서 수행되는 응용 프로그램들은 해당 릴레이션에 접근하기 위해서 타 프로세서와 통신을 하여야 하므로, 릴레이션에 대한 접근 시간이 더 걸린다.

릴레이션의 종류들 중 NOR 릴레이션은 지역 프로세서를 하나만 지정할 수 있다. 이러한 릴레이션을 등록할 경우, 해당 릴레이션을 가장 많이 접근할 프로세서를 지역 프로세서로 지정해야 성능 향상의 효과를 볼 수 있다.

LOR 릴레이션이나 DIS 릴레이션은 여러 개의 지역 프로세서들을 지정할 수 있는 데, 각 지역 프로세서는 서로 다른 튜플 값들을 갖게 된다. 이러한 릴레이션은 각 지역 프로세서의 데이터 등록 시, 수시로 접근되는 튜플들을 등록하여야 성능 향상의 효과를 볼 수 있다.

REP 릴레이션은 중복 릴레이션으로도 불리며, 이 릴레이션은 위에서의 정의와 같이 하나의 릴레이션이 여러 프로세서에 동일하게 저장되는 형태의 릴레이션이다. 그러므로, 이러한 릴레이션은 여러 개의 지역 프로세서들에 등록이 가능하다. 중복 릴레이션을 수시로 검색하는 지역 프로세서들에 등록하게 되면, 자신의 프로세서로부터 데이터를 로컬로 접근하기 때문에 검색이 빠르게 이루어진다. 그러나, 릴레이션 변경 시에는 DBMS가 중복되어 있는 모든 데이터를 변경하여야 하므로, 많은 시간이 소요된다. 따라서, 중복 릴레이션은 검색이 자주 발생하고 변경은 자주 발생하지 않는 데이터를 등록하는 것이 좋다.

3.2 릴레이션의 접근 방법

STAREX 교환기의 DBMS는 다양한 릴레이션 접근 방법(access method)들을 제공한다. 사용자들은 릴레이션의 특성을 잘 고려하여 해당 릴레이션에 가장 적합한 접근 방법을 선택하여야 한다.

교환기 DBMS가 제공하는 접근 방법들에는 sequential, index, binary, index sequential, hash 접근 방법들이 있다. 본 절에서는 각 접근 방법에 대하여 소개하고, 이러한 접근 방법을 효율적으로 사용하는 방법을 소개한다.

3.2.1 Sequential

Sequential 접근 방법은 STAREX 교환기 DBMS가 제공하는 접근 방법들 중 가장 간단한 접근 방법이다. 이 접근 방법은 키 값에 상관없이 튜플들이 순차적으로 저장되며, 데이터 접근 시, 각각의 키 값을 순차적으로 비교함으로써 이루어진다. 이러한 접근 방법은 요청한 튜플이 릴레이션의 n번째 위치에 있을 경우, n번을 비교한 후, 해당 튜플에 접근할 수 있다. 즉, 이러한 접근 방법의 평균 접근 횟수는 (튜플 전체의 개수)/2가 된다.

따라서, 이 접근 방법은 튜플의 크기가 큰 릴레이션에 대하여 적용할 경우, 상당히 많은 시간이 소요될 수 있다. 그러나, 이 접근 방법은 튜플들이 키 값에 관계없이 순차적으로 저장될 수 있으므로, 주기억장치의 크기가 필요한 튜플의 개수만큼만 결정하면 된다. 이러한 접근 방법은 튜플의 개수가 많지 않은 릴레이션에 한하여 사용하는 것이 좋다.

3.2.2 Index

Index 접근 방법은 STAREX 교환기 DBMS가 제공하는 접근 방법들 중 가장 빠른 접근 방법이다. 이 접근 방법은 키 값이 0부터 순차적으로 증가하는 릴레이션에 한하여 사용 가능하며, 원하는 튜플의 주소를 계산하여 해당 튜플을 접근하는 아주 빠른 접근 방법이다. 즉, 키 값이 n인 튜플을 접근하고자 하는 경우에 (릴레이션의 시작 주소) + n * (튜플의 크기)를 계산하여, 단 한번에 해당 튜플의 주소를 알아낼 수 있다. 이러한 접근 방법의 평균 접근 횟수는 1이다.

따라서, 이 접근 방법을 사용할 경우, 상당히 빠른 속도로 데이터 접근을 할 수 있다는 것을 알 수 있다. 그러나, 모든 릴레

이션들이 index 접근 방법으로 등록될 수 있는 것은 아니다. Index 접근 방법은 빠른 검색 속도를 제공하나, 사용 가능한 릴레이션에는 어느 정도의 제약 조건이 따른다. 키 값의 범위에 따라 릴레이션이 차지하는 주기억장치의 크기가 결정되므로, 키의 범위가 그다지 넓지 않거나, 키의 범위가 넓다 하더라도, 거의 모든 키 값에 대하여 해당 튜플이 존재하는 릴레이션이 적합하다.

3.2.3 Binary

Binary 접근 방법[4]은 순서대로 sort된 키 값에 따라 이진 검색을 하는 접근 방법이다. 따라서, 튜플 접근 요청 시, 이진 검색을 수행하게 되므로, 평균 튜플 검색 횟수는 log(전체 튜플의 수)이다.

이 접근 방법은 키 값이 튜플이 저장된 순서대로 순차적으로 증가하면서, index 접근 방법과는 달리, 키 값이 0부터 순차적으로 하나씩 증가되지 않는 릴레이션에 사용하는 것이 적합하다.

3.2.4 Index Sequential

Index sequential 접근 방법은 두 가지 키 값을 사용하는 접근 방법으로서, 첫 번째 키 값을 가지고 index 접근 방법을 수행하여 해당 튜플을 찾은 후, sequential 접근 방법으로 원하는 튜플을 검색하는 방법이다. 이 접근 방법은 index 접근 방법보다는 느리지만, sequential 접근 방법보다는 빠른 접근 방법이라고 할 수 있다.

이러한 접근 방법을 첫 번째 키 값의 범위가 그다지 넓지 않고, 그 키 값에 해당하는 튜플들이 그다지 많지 않은 경우에 사용하는 것이 적합하다.

3.2.5 Hash

일반적으로, hashing 기법은 데이터를 빠르게 처리하기 위한 기법으로 그 동안 많이 연구[5,6,7,8]되어 왔다. STAREX 교환기의 DBMS가 제공하는 hash 접근 방법은 검색을 원하는 튜플의 키 값에 해쉬 함수를 적용하여 해쉬 버켓을 얻고, 버켓이 가리키는 튜플의 linked list를 검색하여 튜플을 찾는 방법이다.

Hash 접근 방법의 장점은 검색해야 할 튜플의 수를 linked list의 길이로 줄여 빠른 접근을 할 수 있다는 것이다. 그러나, 버켓으로 인한 기억 장치가 추가적으로 요구되므로, 이러한 버켓의 수가 릴레이션이 차지하는 기억 장치의 크기에 그다지 영향을 미치지 않는 경우에만 사용하여야 한다. 따라서, hash 접근 방법은 사용자가 사전에 충분히 그 방법을 이해하고 기억장치의 크기 및 link의 수를 확인하여 사용하여야 확실한 효과를 얻을 수 있다.

3.3 실시간 명령어

STAREX 교환기에서는 응용 프로그램들은 프로그래밍 언어로서 CHILL(CCITT High Level Language)을 사용하고 있다. 이러한 응용 프로그램들은 CHILL로 작성한 프로그램에서 데이터베이스에 접근할 수 있도록 CHILL 내장형 데이터 조작어(EDML : CHILL Embedded Database Manipulation Language)를 제공하고 있다. 응용 프로그램 작성자가 DBMS로부터 원하는 데이터를 얻기 위해서는 EDML 처리기가 제공하는 적당한 명령어들을 구문에 맞게 사용하면 된다.

EDML 명령어의 종류로는 제어 명령어(control command), 데이터 접근 명령어(data access command), 통계 명령어(statistical command), 실시간 명령어(real-time command), 트랜잭션 명령어(transaction command)가 있다. 이 중, 실시간 명령어는 데이터 접근 명령어가 해당 릴레이션의 접근 방법에 따라 접근함으로써 소요되는 시간이 응용 프로그램의 실시간 특성을 만족시키지 못하는 문제를 해결하기 위하여 사용자에게 제공하는 명령어로서, 데이터의 특성에 따라 데이터 접근 시간을 극소화시킨 것이다. Index 접근 방법을 갖는 릴레이션에 대해서 적용할 수 있는 이런 명령어들에는 EDML 처리기에서 해당 릴레이션 내 원하는 튜플까지의 상대적인 거리를 컴파일 과정에서 미리 계산함으로써 DBMS에서 소요되는 시간을 극소화시킨 명령어들(DSELECT, DUPDATE, DINSERT, DDELETE)과 데이터베이스 엔진을 거치지 않고 직접 데이터베이스를 접근하여 사용자에게 결과를 알려주는 명령어들(PSELECT, PUPDATE, PINSERT, PDELETE)과 사용자가 DBMS를 전혀 거치지 않고 직접 접근할 수 있도록 릴레이션의 시작 주소를 알려주는 명령어(READ_ADR)가 있다.

표 1은 STAREX 교환기에서 검색 명령어들의 성능을 측정한 결과이다.

<표 1> 검색 명령어들의 성능 비교

검색 명령어	수행 시간(usec)	초당 수행 회수
SELECT	416	2404
DSELECT	132	7576
PSELECT	20.8	48077
READ_ADR을 사용한 검색	3.76	265957

위에서 본 결과와 같이 실시간 명령어를 사용하였을 때, 일반 데이터 접근 명령어들보다 훨씬 좋은 성능을 얻을 수 있다. 일반 데이터 접근 명령어에 비해 DSELECT는 3배, PSELECT는 20배, READ_ADR을 사용한 검색은 100배 이상의 성능을 보이고 있다.

READ_ADR 명령어를 사용하는 경우는 처리가 상당히 빠르게 이루어지지만, 응용 프로그램이 DBMS를 전혀 거치지 않고 직접 데이터를 접근하는 경우이므로, 데이터에 대한 접근에 대한 책임이 응용 프로그램에게 있게 된다. 따라서, READ_ADR은 특수한 경우가 아니면, 사용하지 않아야 한다.

이러한 실시간 명령어들은 항상 사용이 가능한 것은 아니고, 여러 가지 제약이 따른다. 따라서, 응용 프로그램들은 이러한 제약 사항들을 잘 고려한 후, 그 프로그램에 가장 적합한 명령어를 사용하여야 한다.

3.4 데이터 접근 명령어의 옵션

EDML에서 제공하는 명령어들 중 데이터 접근 명령어는 릴레이션의 필요한 튜플을 접근하기 위해서 사용되는 명령어이다. 데이터 접근 명령어에는 튜플을 검색하기 위한 SELECT 명령어, 삭제하기 위한 DELETE 명령어, 갱신하기 위한 UPDATE 명령어, 그리고, 새로운 튜플을 삽입하기 위한 INSERT 명령어가 있다. 이와 같은 명령어들을 구문에 맞게 사용하여 데이터베이스에 접근하게 되면, 릴레이션을 튜플 단위로 접근하게 된다.

이러한 EDML 명령어들은 어떻게 잘 사용하느냐에 따라 DBMS의 접근 성능은 크게 달라진다. 본 절에서는 이러한 명령어들 중 SELECT 명령어와 UPDATE 명령어를 효율적으로 사용하는 방법에 대하여 기술한다.

우선, SELECT 명령어의 syntax는 다음과 같다.

SELECT relation-name [qualification-clause] [options] [DEST := processor_id];

위의 syntax에서 relation-name은 릴레이션 이름을 나타내고, qualification clause는 검색하고자 하는 조건을 기술하도록 되어 있다. options에는 사용자가 필요에 따라서 사용할 수 있는 몇 가지 옵션들이 제공되는 데, 이 중 IN SEQ 옵션은 어떻게 잘 사용하느냐에 따라 DBMS의 접근 시간에 영향을 미친다.

IN SEQ 옵션이란 동일한 조건을 갖는 앞의 SELECT 명령어가 찾은 튜플 이후부터 검색하여 주어진 조건 절을 만족하는 다음 튜플을 읽어 가는 옵션이다. 따라서, IN SEQ 옵션을 사용하기 위해서는 반드시 먼저 SELECT 문을 사용하여야 한다. 마

지막으로 조건을 만족하는 튜플이 더 이상 존재하지 않는 경우에는 "DB_LAST_TUP"가 접근 결과로 사용자에게 전달된다.

UPDATE 명령어의 syntax는 다음과 같다.

UPDATE relation-name [qualification-clause] modification clause [options] [DEST := processor id];

위의 syntax는 SELECT 명령어와 마찬가지로 relation-name은 릴레이션 명을 나타내고, qualification clause는 검색하고자 하는 조건을 기술하도록 되어 있다. modification clause에는 변경하고자 하는 내용을 기술하면 된다. 이러한 방법으로 릴레이션을 변경하게 되면, DBMS는 조건을 만족하는 첫 번째 튜플을 찾아 변경을 한다. options에는 사용자가 필요에 따라서 사용할 수 있는 몇 가지 옵션들이 제공되는 데, 이 중 ALLUP 옵션을 잘 사용하면, DBMS의 접근 성능을 향상시킬 수 있다. ALLUP 옵션은 조건을 만족하는 모든 튜플들을 변경하는 옵션이다.

예를 들어, R_A라는 릴레이션이 index 접근 방법을 사용하고 10000개의 튜플들로 구성되어 있으며, 각 튜플의 키 애트리뷰트인 D_KEY 애트리뷰트의 값이 0부터 9999까지로 구성되어 있다고 가정하자. 이러한 릴레이션에서 D_1 애트리뷰트의 값이 10인 튜플들을 검색하여 D_2 애트리뷰트의 값을 20으로 변경하고자 한다. 이를 CHILL 언어로 코딩하면, 다음과 같이 할 수 있다.

```
do for I := 0 to 9999;
  $ select R_A where (D_KEY = I);
  if (R_A.D_1 = 10) then
    $ update R_A where (D_KEY = I) to (D_2 := 20);
  fi;
od;
```

<예제 프로그램 1>

위와 같은 코딩 방법은 DBMS가 제공하는 장점을 제대로 살리지 못한 비효율적인 프로그램이다. 이러한 방법으로 코딩하면, 실제 D_1 애트리뷰트의 값이 10인 튜플이 얼마나 있는 지에 관계없이 R_A라는 릴레이션을 검색하는 데에만 SELECT 명령어를 10000번 수행하여야 한다. 변경하기 위한 UPDATE 명령어는 실제로 D_1 애트리뷰트의 값이 10인 튜플의 개수만큼 수행하게 된다. 즉, 위의 프로그램은 해당 릴레이션에 접근하기 위하여 EDML 명령어를 최소한 10000번 수행하게 되고, 최대 20000번 수행하게 된다.

이 프로그램의 데이터베이스 접근 회수를 분석해 보자. Index 접근방법은 3.2.2절에서 본 바와 같이 한 EDML 명령어를 수행하는 데, 한 번만 릴레이션을 접근한다. 따라서, 예제 프로그램 1에서 릴레이션에 대한 접근 회수는 명령어의 수행 회수와 동일하므로 최소 10000번에서 최대 20000번 수행된다. 실제로, 교환기 개발자들이 이렇게 비효율적으로 코딩하는 경우가 종종 있다.

위와 동일한 결과를 가져다주는 방법으로 다음과 같이 코딩할 수 있다.

```
$ select R_A where (D_1 = 10);
do while (DB_DB_INFO.ACC_STA = DB_SUCCESS);
  $ update R_A where (D_KEY = R_A.D_KEY)
    to (D_2 := 20);
  $ select R_A where (D_1 = 10) in seq;
od;
```

<예제 프로그램 2>

위와 같은 방법의 프로그래밍은 예제 프로그램 1보다 효율적으로 프로그래밍한 것이다. SELECT 명령어에서 IN SEQ 옵션을 사용하면, 종전에 검색한 튜플 이후부터 검색하면 되므로, 이와 같은 특성을 잘 이용하여 해당 릴레이션을 접근하였다. 이와 같은 방법으로 코딩하면, R_A라는 릴레이션에서 실제 D_1 애트리뷰트의 값이 10인 튜플들의 개수만큼 SELECT 및 UPDATE 명령어가 수행하게 된다. 즉, D_1 애트리뷰트의 값이 10인 튜플이 하나도 없으면, EDML 명령어가 한번만 수행하게 된다. 즉, 위의 프로그램은 EDML 명령어가 최소한 1번 수행하게 되고, 최대 20001번 수행하게 된다.

이 프로그램의 데이터베이스 접근 회수를 분석해 보자. R_A라는 릴레이션은 index 접근 방법을 사용하지만, 이 프로그램에서는 키 애트리뷰트가 아닌 D_1이라는 애트리뷰트를 가지고 해당 릴레이션을 검색하게 되므로, 여기서 검색하는 데 사용되는 접근 방법은 sequential 접근 방법이다. Sequential 접근 방법은 3.2.1절에서 본 바와 같이 각 튜플을 비교하여 이루어지게 되므로, 상당히 많은 접근 회수를 갖는 접근 방법이다. 그러나, 이 프로그램에서는 IN SEQ 옵션을 반복하여 사용하기 때문에 각 EDML 명령어를 실행할 때, 종전에 접근되었던 튜플들은 더 이상 접근되지 않는다. 즉, SELECT 명령어가 몇 번 실행되었던지 관계없이 검색하는 데 필요한 접근 회수는 무조건 튜플의 개수 더하기 1이다. 1을 더하는 이유는 릴레이션의 끝을 확인하기 위하여 한 번 더 검색하기 때문이다. 즉, 검색하는 데 필요한 접근 회수는 무조건 10001이다. UPDATE 명령어는 키 애트리뷰트를 사용하므로, index 접근 방법을 이용하여 접근하게 된다. 즉, 각 UPDATE 명령어에 대한 접근 회수는 1이고, 이 명령어는 D_1 애트리뷰트의 값이 10인 튜플에 대하여 이루어지므로, 해당 릴레이션에 대한 접근이 최소 0에서 최대 10000번 이루어지게 된다. 즉, 총 접근 회수는 최소 10001번에서 최대 20001번 이루어지게 된다.

예제 프로그램 2의 총 접근 회수는 예제 프로그램 1과 유사하지만, EDML 명령어의 수행 회수는 예제 프로그램 2가 적으므로, 예제 프로그램 2가 보다 좋은 성능을 보인다고 할 수 있다.

마지막으로, 위와 동일한 결과를 가져다주는 방법으로 다음과 같이 코딩할 수 있다.

```
$ update R_A where (D_1 = 10) to (D_2 := 20) allup;
```

<예제 프로그램 3>

위와 같은 방법의 프로그래밍은 예제 프로그램 1이나 2보다 훨씬 효율적인 프로그래밍 방법이다. 이 방법은 UPDATE 명령어가 제공하는 옵션의 장점을 잘 살려서 코딩한 방법이다. 이와 같은 방법으로 코딩하면, R_A라는 릴레이션을 변경하기 위하여 EDML 명령어가 무조건 한 번만 수행하게 되는 아주 효율적인 방법이다.

이 프로그램의 데이터베이스 접근 회수를 분석해 보자. 이 프로그램은 키 애트리뷰트가 아닌 D_1이란 애트리뷰트를 사용하여 접근하므로 sequential 접근 방법을 사용하고 있는 것이다. D_1의 값이 10인 튜플들을 찾기 위해서는 모든 튜플들을 접근하여야 하므로, 총 접근 회수는 10000이다.

따라서, 예제 프로그램 3은 예제 프로그램 1이나 2보다 EDML 명령어의 총 수행 회수와 릴레이션에 대한 총 접근 회수가 적은 아주 효율적인 프로그램이라고 할 수 있다.

표 2는 각 예제 프로그램에 대하여 EDML 명령어의 수행 회수를 비교한 표이고, 표 3은 각 예제 프로그램에 대하여 데이터베이스의 접근 회수를 비교한 표이다.

<표 2> EDML 명령어의 수행 회수

<표 3> 데이터베이스의 접근 회수의 비교

지금까지 본 바와 같이 동일한 프로그램을 코딩하더라도 어떠한 방법으로 코딩하느냐에 따라 DBMS의 접근 성능은 크게 좌우될 수 있다. 따라서, EDML을 사용하더라도 무조건 원하는 결과를 얻는 것이 중요한 것이 아니라, 어떠한 방법으로 코딩하여야 보다 효율적인 DBMS의 접근 성능을 향상시킬 수 있는가를 잘 고려하여 코딩하여야 한다.

IV. 결론

STAREX 교환기의 DBMS는 교환기 시스템에서 동시 다발적으로 발생하는 데이터 요구를 실시간으로 처리할 수 있어야 한다. 호처리 과정에서 필요한 데이터 접근 요구의 실시간 처리는 필수적이라고 할 수 있다. 따라서, 교환기 DBMS는 이러한

예제 프로그램	Best Case (회수)	Worst Case (회수)
1	10000	20000
2	1	20001
3	1	1

실시간 처리를 하기 위하여 여러 가지 기능들을 제공한다. 그러나, 응용 프로그래머들은 자신의 결과를 얻기에만 급급하여 이러한 기능들을 마음대로 사용하면, 원하는 결과를 얻을 수는 있

예제 프로그램	Best Case (회수)	Worst Case (회수)
1	10000	20000
2	10001	20001
3	10000	10000

겠지만, DBMS의 접근 성능은 보장받을 수가 없다. 이 논문에서는 STAREX 교환기 DBMS가 제공하는 여러 가지 실시간 처리 지원 기능과 그 효율적인 사용 기법을 기술하였다. 응용 프로그램들이 이러한 DBMS의 기능들을 잘 이해하고 효율적으로 사용할 경우, 교환기 전체의 성능 향상이 이루어질 수 있을 것이다.

V. Reference

[1] Garcia-Molina, H., "Main Memory Database Systems : An Overview," IEEE Trans. on Knowledge and Data Engineering, Vol. 4, No. 6, pp. 509-516, 1992.

[2] Codd, E. F., "Further Normalization of the Database Relational Model," Data Base Systems, pp. 33-64, 1972.

[3] Codd, E. F., "Relational Completeness of Data Base Sublanguages," pp. 65-98, 1972.

[4] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., "Sorting Information in Files," Data Structures and Algorithms, pp. 361-374, 1985.

[5] Litwin, W., "Linear Hashing : A New Tool for File and Table Addressing," Proc. 6th Intl. Conf. Very Large Data Bases, pp. 212-223, 1980.

[6] Larson, P., "Dynamic Hash Tables," Comm. of ACM, Vol.31, No.4, pp. 446-457, 1988.

[7] Fagin, R., Nievergelt, J., Pippenger, N., and Strong, H. R., "Extendible Hashing : A Fast Access Method for Dynamic Files," ACM Trans. On Database Systems, Vol.4, No.3, pp. 315-344, 1979.

[8] Knuth, D., "The Art of Programming-Sorting and Searching," Addison-Wesley Publishing Co. Inc., 1973.