위치기반 서비스를 위한 다중레벨 DBMS에서 질의 분류 컴포넌트의 설계 및 구현

장 석 규[†] · 어 상 훈[††] · 김 명 근[†††] · 배 해 영[††††]

요 약

현재 위치기반 서비스를 제공하기 위하여 다양한 시스템들이 사용되고 있다. 그러나 기존의 시스템들은 상당히 많은 사용자들에게 빠른 서비스를 제공하기에는 적합하지가 않다. 이러한 문제점을 해결하기 위하여 빠른 데이터 처리와 대용량의 데이터 관리를 동시에 지원하는 다중레벨 DBMS를 사용하여야 한다. 스냅샷을 갖는 다중레벨 DBMS는 디스크에 모든 데이터를 가지고 있으며, 빠른 처리를 요구하는 데이터는 스냅샷의 형태로 메인메모리 데이터베이스에서 관리한다. 이 시스템의 성능을 최적화하여 위치기반 서비스를 제공하기 위해서는 스냅샷에 존재하는 데이터를 효율적으로 사용할 수 있도록 질의를 분류하는 컴포넌트가 필요하다. 본 논문에서는 위치기반 서비스를 위한 다중레벨 DBMS에서 질의 분류 컴포넌트를 설계하고 구현한다. 제안된 컴포넌트는 입력된 질의를 메모리 질의, 디스크 질의, 하이브리드 질의로 분류하여 스냅샷 사용율을 높이고, 스냅샷의 일부분을 사용할 수 있도록 질의의 비공간과 공간 필터 조건을 분할하는 메커니즘을 사용하였다. 따라서, 제안된 컴포넌트는 효율적인 질의 분류를 통하여 스냅샷을 최대한 이용함으로써 시스템의 성능을 향상시킨다.

키워드 : 질의 분류, 위치기반 서비스, 다중레벨 DBMS

Design and Implementation of Query Classification Component in Multi-Level DBMS for Location Based Service

Seok-Kyu Jang[†] · Sang Hun Eo[††] · Myung-Keun Kim[†††] · Hae-Young Bae[††††]

ABSTRACT

Various systems are used to provide the location based services. But, the existing systems have some problems which have difficulties in dealing with faster services for above million people. In order to solve it, a multi-level DBMS which supports both fast data processing and large data management support should be used. The multi-level DBMS with snapshots has all the data existing in disk database and the data which are required to be processed for fast processing are managed in main memory database as snapshots. To optimize performance of this system for location based services, the query classification component which classifies the queries for efficient snapshot usage is needed. In this paper, the query classification component in multi-level DBMS for location based services is designed and implemented. The proposed component classifies queries into three types: (1) memory query, (2) disk query, (3) hybrid query, and increases the rate of snapshot usage. In addition, it applies division mechanisms which divide aspatial and spatial filter condition for partial snapshot usage. Hence, the proposed component enhances system performance by maximizing the usage of snapshot as a result of the efficient query classification.

Key Words : Query Classification, Location Based Service, Multi-Level DBMS

1. 서 론

오늘날 무선 인터넷과 모바일 컴퓨팅 기술이 크게 발전하여, 휴대전화, PDA(Personal Digital Assistant)와 같은 모바일 장치의 사용이 급격하게 증가하였으며 보편화되어 있다. 또한, 사용자의 위치 정보를 이용한 트래킹, 친구찾기와 같은 위치 기반 서비스(Location Based Service)들이 다양하게 제공되고 있다. 이러한 환경에서 수많은 사용자들에게 동시에 위치 기반 서비스를 제공하기 위해서는 상당히 많은 트랜잭션을 빠르게 처리하는 데이터베이스 시스템이 필요하다[1, 2].

현재, 이러한 대용량의 위치 데이터의 처리가 요구되는 LBS를 지원하기 위해서 시공간 데이터베이스[3, 4]나 이동객체 데이터베이스[5, 6]와 같은 시스템을 클러스터로 구성

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구 센터 육성·지원사업의 연구 결과로 수행되었음.
† 준 회 원 : 인하대학교 대학원 컴퓨터·정보공학과 석사과정
†† 준 회 원 : 인하대학교 대학원 컴퓨터·정보공학과 통합과정
††† 준 회 원 : 인하대학교 대학원 컴퓨터·정보공학과 박사과정
†††† 종신회원 : 인하대학교 컴퓨터공학부 교수
논문접수 : 2005년 6월 8일, 심사완료 : 2005년 7월 25일

하여 사용하고 있다. 그러나 이와 같은 시스템은 네트워크를 이용한 통신방법에 기반하고 있으며, 네트워크 장애에 대한 근본적인 대책이나 네트워크 비용이 고려되어야만 한다[7]. 이때, 클러스터를 구성하고 있는 노드의 개수를 줄이게 되면 네트워크에 대한 부하를 상당히 줄일 수가 있다. 따라서, 시스템을 구성하고 있는 각각의 데이터베이스 시스템이 더욱 빠른 트랜잭션 처리를 지원하고 대용량의 데이터를 관리할 수 있다면, 더 적은 노드로 이루어진 클러스터 시스템으로 대용량 서비스를 지원할 수가 있다. 즉, 클러스터 시스템을 구성하고 있는 각각의 노드는 고성능의 다중레벨 DBMS로 대체되어야만 한다.

다중레벨 DBMS는 접근 속도와 저장 용량이 서로 다른 여러 개의 저장장치를 사용하는 DBMS로서 빠른 처리와 대용량 관리의 요구사항을 만족시킨다[8]. 이 시스템의 메인 메모리 데이터베이스에는 매우 빠른 연산을 요구하는 데이터들이 관리되고 있으며, 방대한 양의 나머지 데이터들은 디스크 데이터베이스에서 안정적으로 관리된다. 따라서 다중레벨 DBMS는 데이터베이스 클러스터 시스템에서 최상의 성능을 가지는 단일 노드로서 적합하다. 그러나, 일반적인 다중레벨 DBMS는 서로 다른 레벨의 데이터베이스에 저장되어 있는 데이터들이 서로 독립적이어서, 하위 레벨의 데이터에 접근하기 위해서는 상위레벨의 버퍼 공간으로 이동해서 사용해야만 한다. 즉, 디스크데이터베이스에 저장되어 있는 데이터는 반드시 메모리영역의 버퍼 공간으로 읽어와야만 하는 레벨간의 I/O비용이 상당히 큰 문제점이 있다[8].

이러한 문제점은 메인메모리 데이터베이스에 빠른 검색을 요구하는 데이터들을 미리 스냅샷으로 생성하여 사용함으로써 해결할 수 있다. 모든 데이터들은 디스크 데이터베이스에 존재하며, 테이블 단위의 검색 질의 프리디킷을 사용하여 스냅샷을 생성한다. 메인메모리 데이터베이스에 생성된 스냅샷을 이용하면 I/O비용 없는 매우 빠른 처리로 상당히 많은 트랜잭션 처리를 지원할 수 있다. 그러나, 이 시스템에서는 질의가 요구하는 데이터가 스냅샷에 존재하는 지의 여부를 판단하여 디스크 질의, 메모리 질의, 하이브리드 질의의 세 가지로 나누어서 처리 하여야만 한다. 또한, 요구되는 데이터의 일부만 스냅샷에 존재할 경우에 따른 처리도 고려되어야만 한다[9].

본 논문에서는 위치기반 서비스를 위한 다중레벨 DBMS에서 질의 분류 컴포넌트를 설계하고 구현한다. 제안된 컴포넌트는 메타데이터와 질의를 분류하는 알고리즘들로 구성된다. 질의 분류 알고리즘은 최종적으로 질의 종류를 결정하는 알고리즘, 실행계획 순회 알고리즘, 사용될 테이블의 위치를 결정하는 알고리즘, 사용되는 필드를 분할하는 알고리즘, 비공간과 공간 조건을 분할하는 알고리즘, 분할된 새로운 프리디킷을 생성하는 알고리즘들로 이루어진다. 질의 분류의 컴포넌트의 특징은 최대한 스냅샷을 사용하게 질의를 분류하며, 특히 하이브리드 질의의 경우 스냅샷의 일부 데이터만 요구되는 경우에도 해당 데이터를 사용할 수 있게 질의를 분류한다. 또한 제안된 컴포넌트를 구현하기 위하여 위치기반 서비스 환경을 구축하고, 관리자 도구를 이용하여 모의 실험을 수행한다. 분류된 질의 간의 성능 비교를 통해 질의 분류가 시스템의 성능에 미치는 영향을 확인하고 평가한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련연구로 멀티 데이터베이스 시스템에서의 질의 분류와 다중레벨 DBMS를 소개하고, 3장에서는 본 논문의 환경인 스냅샷을 갖는 다중레벨 DBMS를 설명한다. 4장의 질의 분류 컴포넌트의 설계에서는 질의 종류, 메타데이터, 질의 분류 알고리즘을 기술하고, 5장에서는 구현 및 평가를 하며 6장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 멀티 데이터베이스 시스템 (MDBS: Multi-database system)을 위한 질의 분류를 소개하고 다중레벨 DBMS를 설명한다.

2.1 멀티 데이터베이스를 위한 질의 분류

MDBS는 분산 환경에서 이기종의 데이터베이스 관리 시스템에 의해 관리되고 있는 기존의 자치적인 로컬 데이터베이스들로부터 데이터를 통합하여 사용한다. MDBS와 분산 데이터베이스 시스템(DDBS: Distributed Database System)과의 가장 큰 차이점은 바로 지역 자치성이다[10].

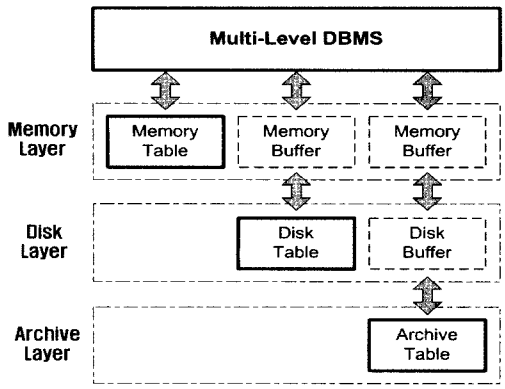
질의 분류는 지역적인 자치성을 가지고 있는 MDBS에서 지역 성능에 의존적이지 않은 전역 질의 최적화를 위해 사용되었다. 이 방법에서는 각각의 질의를 크게 클러스터된 인덱스 질의, 클러스터 되지 않은 인덱스 질의, 비 인덱스 질의의 크게 세 가지로 나누어 샘플링 하여 비용을 계산하였다[11, 12, 13].

그러나 독립된 시스템의 개념으로 볼 때 단일 노드로 쓰이는 스냅샷을 갖는 다중레벨 DBMS에서는 다른 방식의 질의 분류가 필요하다. 다시 말하면, 레벨간의 성능차이와 각각에 저장된 데이터의 특징을 이용하여 시스템을 최적화할 수 있는 방향으로 질의가 분류되어야 한다.

2.2 다중레벨 DBMS

(그림 1)의 다중레벨 DBMS는 세 개의 레벨을 가진다. 최상위 메모리 레이어는 주기억장치인 메인 메모리를 사용하며 하위 레이어인 디스크 레이어와 아카이브 레이어의 버퍼 영역으로 사용된다. 가운데의 디스크 레이어는 디스크 기반의 보조기억 장치를 사용하며 아카이브 레이어의 버퍼 영역으로 사용된다. 최하위의 아카이브 레이어는 값싼 비용으로 대용량을 실현할 수 있으며 테이프, 쥬크박스, 광 디스크 등의 장치를 사용한다[8].

이 시스템은 각각의 레이어 데이터베이스에 저장된 테이블이 독립적인 특징을 가지고 있어, 질의 처리시 단일 테이블의 경우 각각 디스크 테이블을 참조하면 디스크 테이블을, 메모리 테이블을 참조하면 메모리 테이블을 사용하며, 두개 이상의 테이블의 경우에는 메모리와 디스크의 테이블에 대



(그림 1) 다중레벨 DBMS

하여 함께 처리된다. 그러나, 스냅샷을 갖는 다중레벨 DBMS와는 달리 단일 테이블의 일부분이 메모리에 존재하는 경우는 없기 때문에 이에 대한 질의 처리 부분이 고려되지 않았다.

또한 하위레벨의 데이터에 접근하기 위해서는 반드시 상위레벨의 버퍼공간으로 이동하는 I/O비용이 추가적으로 발생하며, 메모리 레이어에 존재하는 테이블들에 대하여 반드시 회복에 대한 고려를 해주어야 함으로 스냅샷을 갖는 다중레벨 DBMS 구조에 비하여 더 큰 부하를 갖고 있다는 단점이 있다. 따라서 잦은 갱신이 일어나는 LBS에는 적합하지 않다.

3. 스냅샷을 갖는 다중레벨 DBMS

본 논문의 환경인 (그림 2)의 스냅샷을 갖는 다중레벨 DBMS는 디스크와 메모리 기반의 데이터베이스를 가지고 있으며 기존의 다중레벨 DBMS와 유사한 구조를 갖는다[14].

모든 데이터는 하위레벨인 디스크 레벨이 존재하며, 빠른 처리를 요구하는 데이터는 상위레벨의 메모리 레이어에 스냅샷의 형태로 존재한다는 차이가 있다. 즉, 각각의 레이어의 데이터가 독립적이지 않다. 또한, 디스크 기반의 데이터베이스를 백업 저장장치로 사용하기 때문에 상위 레이어에 저장된 스냅샷 처리의 경우 데이터 회복의 문제를 고려하지 않은 메인메모리 알고리즘을 사용할 수 있다는 장점이 있다.

<표 1>에서는 다중레벨 DBMS와 스냅샷을 갖는 다중레벨 DBMS를 데이터 처리위치에 따라 비교하였다. 두개의 시스템은 디스크 데이터베이스에서 데이터를 처리하는 경우에는 같은 처리과정을 거치기 때문에 동일한 비용이 드는 것을 알 수가 있다. 그러나 메모리 데이터베이스에서 처리하는 경우에는 회복 메커니즘 처리의 차이점으로 다중레벨 DBMS가 더 많은 부하를 가지고 있다.

〈표 1〉 다중레벨 DBMS 간의 비교

종류 \ 처리위치	메모리 (Database)	디스크 (Database)
다중레벨 DBMS	회복을 고려한 메커니즘 (잦은 로깅)	메모리 버퍼로 이동 후 처리
다중레벨 DBMS (with Snapshot)	회복을 고려하지 않은 메커니즘(압축 동기화)	메모리 버퍼로 이동 후 처리

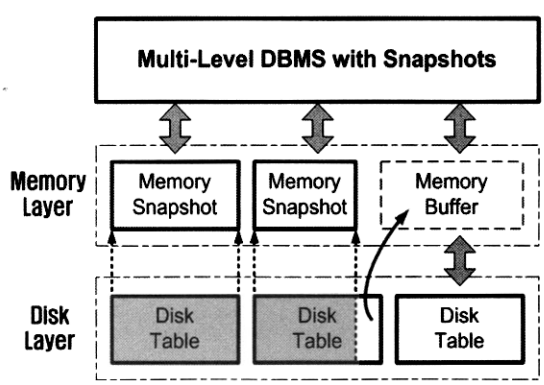
그러나 스냅샷을 갖는 다중레벨 시스템은 두 가지 문제점을 가진다. 첫째, 두개 레벨의 데이터베이스에 데이터를 이중화하여 저장하기 때문에 데이터의 일관성 문제가 발생한다. 둘째, 메인메모리 레벨에서 갱신된 데이터에 대하여 디스크 레벨로 전송하는 과정에서 병목현상이 발생한다.

위에서 언급한 문제점들은 다음과 같은 방법으로 해결한다. 먼저 일관성의 문제를 해결하기 위하여 지연갱신전파(lazy propagation) 방법을 사용한다. 트래킹이나 친구찾기 같은 LBS서비스를 위한 데이터는 특성상 약간의 손실은 상관이 없기 때문에 완화된 일관성을 보장하지만 빠른 응답시간을 보장할 수 있어야 하기 때문이다.

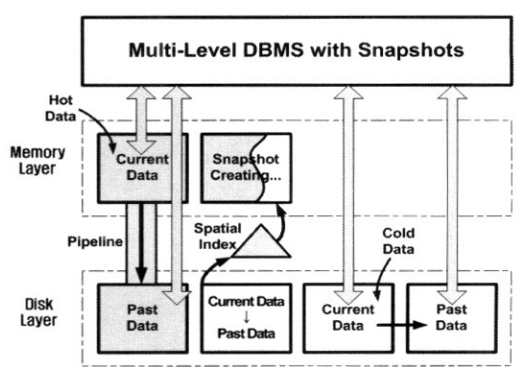
병목 현상은 메모리 레벨에서 디스크 레벨의 처리한계를 넘는 갱신된 데이터를 전송할 때 발생한다. 이 문제를 해결하기 위해서는 (그림 3)에서와 같이 큐로 이루어진 파이프라인을 레벨 간에 설치하고, 파이프라인에 들어온 데이터를 압축하는 메커니즘을 적용하여 생성된 벡터성분을 디스크에 저장하여 해결한다.

또한 (그림 3)에서와 같이 빠른 처리를 요구하는 Hot 데이터에 대해서는 메모리에서 스냅샷으로 현재 데이터를 관리하며, 과거데이터는 디스크에서 관리한다. 그러나 빠른 처리가 아닌 대용량에 초점을 둔 Cold 데이터는 현재 데이터와 과거 데이터를 모두 디스크에서 관리한다.

스냅샷이 생성될 때에는 디스크에서 관리되고 있는 현재



(그림 2) 스냅샷을 갖는 다중레벨 DBMS



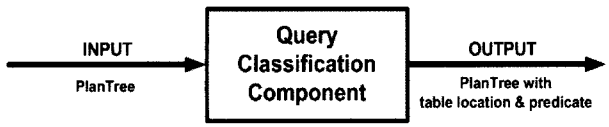
(그림 3) LBS를 위한 스냅샷을 갖는 다중레벨 DBMS

데이터를 이용하며, 생성 중에는 이동객체를 관리하기 위한 스냅샷 용 인덱스가 함께 메모리에 생성된다. 스냅샷으로 생성되는 데이터는 프리디킷을 이용한 단일 테이블의 전체 영역 또는 일부분 영역으로 공간데이터와 비공간 데이터를 함께 포함한다.

4. 질의 분류 컴포넌트의 설계

스냅샷을 갖는 다중레벨 DBMS에서는 모든 데이터가 디스크에 존재한다는 특징 때문에 기존의 다중레벨 DBMS와는 달리 메모리 데이터베이스에 저장되어 있는 데이터를 효율적으로 사용하여야 한다. 따라서, 사용되는 테이블의 위치에 따라 효율적으로 질의를 분류하는 메커니즘이 반드시 필요하다.

본 장에서는 효율적인 질의 분류를 수행하는 질의 분류 컴포넌트를 설계하기 위하여 분류 되어야할 질의 종류와 메타데이터를 정의하며, 핵심 알고리즘인 질의 분류 알고리즘을 기술한다. 질의 분류 컴포넌트는 (그림 4)에서와 같이 질의 처리에 사용되는 실행계획이 입력되며, 실행계획과 함께 데이터가 참조되어야 할 테이블별 위치와 해당 프리디킷을 출력한다. 이 결과는 각 저장관리자에 넘겨져서 해당 데이터를 처리하게 된다.



(그림 4) 질의분류 컴포넌트의 입력과 출력

4.1 질의 종류

질의 분류 컴포넌트에서는 사용되는 데이터의 위치에 따라 다음과 같이 세 가지로 질의를 분류한다.

첫째, 메모리 질의(TQ_MM)는 요구되는 데이터가 모두 스냅샷의 생성조건을 만족하여 메모리에 존재하는 데이터를 사용하는 경우의 질의이다. 둘째, 디스크 질의(TQ_DK)는 요구되는 데이터가 모두 스냅샷의 생성조건에 만족하지 않아 디스크에 존재하는 데이터를 사용하는 경우의 질의이다. 셋째, 하이브리드 질의(TQ_HB)는 요구되는 데이터의 일부만이 스냅샷의 생성조건을 만족하여 메모리에 존재하는 데이터와 디스크에 존재하는 데이터를 동시에 사용하는 질의이다.

<표 2>에서는 테이블 위치에 따른 질의 종류를 보여주고 있다. 질의를 처리하는 경우 실행계획이 단일테이블에 존재하는 데이터를 요구하는 경우와 2개 이상의 멀티 테이블에 존재하는 데이터를 요구하는 경우로 나누어 볼 수가 있다. 메모리 질의와 디스크 질의는 각각 요구되는 데이터의 테이블 위치가 메모리 또는 디스크에 위치해야만 한다. 그러나 하이브리드 질의의 경우 단일테이블에서는 메모리에 일부, 디스크에 일부만 존재하는 것을 고려할 수 있으며, 멀티테이블에서는 질의가 요구하는 테이블들의 개수에 따라 경우의 수가 다양하게 존재할 수 있다.

4.2 메타데이터

메타데이터는 디스크 테이블의 정보와 메모리 테이블의 생성 정보 등을 수록하고 있다. 질의 분류시 사용되는 메타데이터의 스키마는 (그림 5)와 같다.

TBL_DK_ Name	TBL_DK_ OID	TBL_MM_ OID	FieldList	Filter	
				aspatial	spatial

(그림 5) 메타데이터의 스키마

*TBL_DK_Name*은 메모리로 로딩된 스냅샷 테이블의 기반 디스크 테이블 이름이며, 사용자로부터 입력된 질의의 실행계획을 구성하고 있는 테이블들의 이름과 비교하여 해당 테이블을 기반으로 생성된 스냅샷의 존재여부를 판단한다.

TBL_DK_Name 테이블의 메모리 스냅샷 존재 여부가 판단되었다 해도 해당 테이블의 모든 레코드가 메모리에 존재함을 보장할 수 없기 때문에 FieldList와 Filter를 추가적으로 비교분석 한다.

*TBL_DK_OID*와 *TBL_MM_OID*는 각각의 저장관리자에서 테이블을 관리하기 위한 Object Identifier이다. 질의에서 요구하는 테이블의 위치가 결정되었을 때 OID를 이용하여 해당하는 저장관리자에 데이터를 요청한다.

*FieldList*는 스냅샷 테이블의 생성시 적용된 필드의 목록을 가지고 있다. 이것은 레벨 결정 알고리즘에서 필드목록의 포함조건을 분석할 때 사용된다.

마지막으로 *Filter*는 스냅샷 생성시 적용되었던 해당 테이블의 비공간데이터 조건인 *Aspatial Filter*와 공간데이터 조건인 *Spatial Filter*를 포함하고 있다.

4.3 질의 분류 알고리즘

본 절에서는 질의를 분류하는 여러 가지의 알고리즘을 각

〈표 2〉 테이블 위치에 따른 질의 종류

테이블위치	단일테이블(1개)			멀티테이블(2개)					
MM	✓			✓T1 ✓T2		✓T1	✓T1		
DK		✓			✓T1 ✓T2	✓T2		✓T1	
DK+MM			✓				✓T2	✓T2	✓T1 ✓T2
질의종류	TQ_MM	TQ_DK	TQ_HB	TQ_MM	TQ_DK	TQ_HB			

기능별로 살펴본다.

4.3.1 질의의 종류 결정

(그림 6)의 QueryDecision 알고리즘은 실행계획을 입력받아 질의의 종류를 반환한다. 이 알고리즘에서는 실행계획을 순회하면서 스택에 삽입되어있는 테이블별 위치를 꺼내어 분석함으로써 질의의 종류를 결정하는 역할을 한다. 스택 안에는 테이블별 위치와 함께 해당되는 프리디킷을 가지고 있다. 따라서, 질의 종류가 결정됨과 동시에 각 테이블의 해당 프리디킷을 통해 원하는 데이터를 각각 메모리 또는 디스크를 통해 가져올 수 있게 된다.

```
TypeQuery QueryDecision (Plantree p)
// p = plantree pointer
Begin
  Stack s // Stack containing table locations & predicates
  Traverse(p, s)
  while !empty(s)
      Location = pop(s)
      if ( ALL(Locations) == TL_MM )
        return TQ_MM
      else if ( ALL(Locations) == TL_DK )
        return TQ_DK
      else
        return TQ_HB
      endif
  endwhile
end.
```

(그림 6) 질의 결정 알고리즘

4.3.2 실행 계획의 순회

(그림 7)의 Traverse 알고리즘은 실행계획과 함께 결과를 저장하기 위한 스택을 입력받는다. 실행 계획을 후위순회하면서 테이블 노드를 만날 경우 레벨을 결정하는 함수를 호출한다. 순회가 끝나면, 모든 테이블들의 위치와 함께 프리디킷 정보를 스택에 넣은 다음 종료한다.

```
void Traverse (Plantree p, Stack s)
Begin
   if ( p != NULL )
      Traverse ( LeftChildNode(p), s )
      Traverse ( RightChildNode(p), s )
      if ( IsTableNode(p) )
         {[p],<p>} = LevelDecision (p)
         // [x] = location of x, <x> = predicate of x
         push(s, {[p],<p>})
      endif
   endif
end.
```

(그림 7) 실행 계획의 순회 알고리즘

4.3.3 테이블의 레벨 결정

(그림 8)의 LevelDecision 알고리즘은 실행계획의 하나의 노드인 테이블을 입력받아 해당되는 테이블의 위치와 함께 프리디킷을 결정하는 역할을 한다.

첫 번째 조건문에서 메타데이터에 테이블의 이름이 존재하는지를 확인한 다음, 두 번째 조건문에서 실행계획에서 요구하는 필드의 목록이 존재하는지를 확인한다. 세 번째와 네 번째 조건문에서 실행계획이 요구하는 데이터의 비공간과 공간 영역을 메타데이터에 수록되어 있는 조건에 포함되는지 판단한다. 따라서, 네 개의 조건을 모두 만족하는 경우 연산에 필요한 데이터를 메모리 스냅샷에서 가져올 수 있다고 판단하여 TL_MM과 함께 요구된 원래의 프리디킷을 반환한다. 하지만 네 개의 조건들 중에 하나라도 만족하지 못하는 경우에는 TL_DK와 함께 요구된 원래의 프리디킷을 반환한다.

MetaDataContains(FieldListOf(t))는 질의가 요구하는 필드 목록이 스냅샷에 존재하는지를 판단하기 위한 방법이다. (그림 9)와 같이 디스크 테이블, 스냅샷 테이블, 질의의 필드 목록을 각각 T, S, Q의 집합으로 정의할 때, Q⊂S를 만족하는 경우 참을 반환한다.

```
{Location, Predicate}[] LevelDecision (Table t)
// t = table pointer
Begin
  if ( MetaDataContains(NameOf(t)) )
    if ( MetaDataContains(FieldListOf(t)) )
      if ( MetaDataContains(AspatialFilterOf(t)) )
        if ( MetaDataContains(SpatialFilterOf(t)) )
          return {TL_MM, original predicate}
        else
          SpatialDivide(); MatchPredicate();
          return {{TL_MM, MMPdk}, {TL_DK, DKPdk}}
        endif
      else
        if ( MetaDataContains(SpatialFilterOf(t)) )
          AspatialDivide(); MatchPredicate();
          return {{TL_MM, MMPdk} {TL_DK, DKPdk}}
        else
          AspatialDivide(); SpatialDivide(); MatchPredicate();
          return {{TL_MM, MMPdk}, {TL_DK, DKPdk}}
        endif
      endif
    else
      if ( MetaDataContains(AspatialFilterOf(t)) )
        if ( MetaDataContains(SpatialFilterOf(t)) )
          FieldDivide(); MatchPredicate();
          return {{TL_MM, MMPdk}, {TL_DK, DKPdk}}
        else
          FieldDivide(); SpatialDivide(); MatchPredicate();
          return {{TL_MM, MMPdk}, {TL_DK, DKPdk}}
        endif
      else
        if ( MetaDataContains(SpatialFilterOf(t)) )
          FieldDivide(); AspatialDivide(); MatchPredicate();
          return {{TL_MM, MMPdk}, {TL_DK, DKPdk}}
        else
          FieldDivide(); AspatialDivide();
          SpatialDivide(); MatchPredicate();
          return {{TL_MM, MMPdk}, {TL_DK, DKPdk}}
        endif
      endif
    endif
  else
    return {TL_DK, original predicate}
  endif
end.
```

(그림 8) 레벨 결정 알고리즘

$$S \subset T, Q \subset T$$

$$T = \{f_1^d, f_2^d, \cdots, f_l^d\}$$

$$S = \{f_1^s, f_2^s, \cdots, f_m^s\}$$

$$Q = \{f_1^q, f_2^q, \cdots, f_n^q\}$$

T : 각각의 필드를 원소로 포함하는 디스크 테이블
S : 각각의 필드를 원소로 포함하는 메모리 스냅샷 테이블
Q : 질의에서 요구하는 필드 집합

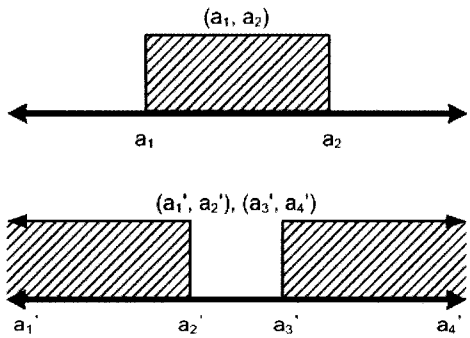
(그림 9) 데이터의 집합 관계 표현

MetaDataContains(AspatialFilterOf(t))는 질의에 포함되어 있는 비공간 데이터에 대하여 각각의 필드에 대한 필터 조건에 대해 스냅샷 생성의 필터조건을 만족하는지를 판단하는 방법이며 아래의 절차를 따른다.

① 스냅샷의 생성조건을 가지는 CNF 필터조건을 식(1)을 이용하여 (그림 10)과 같은 각 필드별 자료구조를 구축한다.

$$C(f_i) = (a_1 \lhd x \lhd a_2) \vee (a_3 \lhd x \lhd a_4) \vee \cdots \vee (a_n \lhd x \lhd a_{n+1})$$
$$a_i \in \{Varible, -\infty, +\infty\}, \quad \lhd \in \{<, \leq, =\} \qquad (1)$$

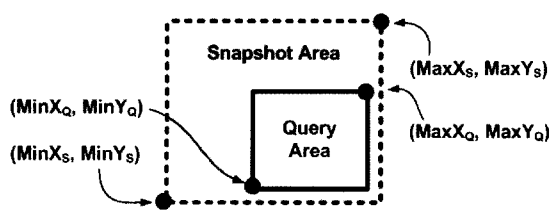
② 위와 같은 방법을 이용하여 질의로부터 입력된 CNF 필터조건을 각 필드별 자료구조로 구축한다.



(그림 10) 비공간 필터조건의 범위 표현

③ 조건의 포함관계를 알아내기 위하여 ①에서 구한 조건 범위와 ②에서 구한 조건 범위를 각 필드별로 비교한다. ①⊃②를 만족할 경우 참을 반환한다.

MetaDataContains(SpatialFilterOf(t))는 질의에 포함되어 있는 공간 데이터에 대하여 영역이 스냅샷 생성 영역에 포함되는 지를 (그림 11)과 같이 판단하는 방법이며 간단한 MBR 비교 연산으로 이루어져 있다.



(그림 11) 스냅샷 영역과 질의 영역의 MBR의 예

4.3.4 필드 분할

(그림 12)의 FieldDivide 알고리즘은 질의가 요구하는 필드목록과 스냅샷의 필드 목록을 가지고 메모리에서 처리할 수 있는 부분과 디스크에서 처리할 수 있는 부분을 나눈다. 메모리 영역은 질의 필드목록과 스냅샷 필드목록의 교집합이 되며, 디스크 영역은 질의가 요구하는 필드목록에서 스냅샷의 필드목록을 뺀 것이 된다.

```
{Fieldlist, Fieldlist} FieldDivide (FieldList Q, FieldList S)
// Q = Query FieldList
// S = Snapshot FieldList
Begin
  MMFieldList =  Q AND S;
  DKFieldList = Q - S;
  return {MMFieldList, DKFieldList}
end.
```

(그림 12) 필드 분할 알고리즘

4.3.5 비공간 조건 분할

(그림 13)의 AspatialDivide 알고리즘은 질의가 요구하는 비공간 필터 조건과 스냅샷이 가지고 있는 비공간 필터조건을 가지고 메모리에서 처리할 수 있는 조건과 디스크에서 처리할 수 있는 조건으로 나눈다. 메모리 영역은 질의 필터조건과 스냅샷 필터조건의 교집합이 되며, 디스크 영역은 질의가 요구하는 필터조건에서 스냅샷의 필터조건을 뺀 것이 된다.

```
{AFilter, Afilter} AspatialDivide
                  (AFilter QCondition, AFilter SCondition)
// AFilter = Aspatial Filter
Begin
  MMAFilter = QCondition AND SCondition;
  DKAFilter = QCondition - SCondition;
  return {MMAFilter,DKAFilter}
end.
```

(그림 13) 비공간 조건 분할 알고리즘

4.3.6 공간 조건 분할

(그림 14)의 SpatialDivide 알고리즘은 질의가 요구하는 공간 영역과 스냅샷의 공간 영역을 가지고 메모리에서 처리할 수 있는 부분과 디스크에서 처리할 수 있는 부분을 나눈다. 메모리의 영역은 질의의 영역과 스냅샷 영역이 되며, 디스크 영역은 질의가 요구하는 영역에서 스냅샷의 영역을 뺀 것이 된다.

```
{SFilter, SFilter} SpatialDivide
                  (SFilter QCondition, SFilter SCondition)
// SFilter = Spatial Filter
Begin
  MMSFilter = QCondition AND SCondition;
  DKSFilter = QCondition - SCondition;
  return {MMSFilter,DKSFilter}
end.
```

(그림 14) 공간 조건 분할 알고리즘

4.3.7 분할된 프리디킷의 생성

(그림 15)의 MatchPredicate 알고리즘은 메모리와 디스크를 사용하는 각각의 프리디킷을 설정한다. 레벨결정 알고리즘에서 판단된 8가지의 경우의 수중에 모든 조건이 만족한 경우를 제외한 7가지의 경우의 수에 대하여 각각 디스크와 메모리 별로, 필드목록, 비공간 필터, 공간 필터를 조합하여 새로운 프리디킷을 만든다.

```
{Predicate, Predicate} MatchPredicate (FieldList MMFld, FieldList
DKFld, AFilter MMAFlt, AFilter DKAFlt, SFilter MMSFlt,
SFilter DKSFlt)
Begin
// Fld = original fieldlist
// AFlt = original aspatial filter
// SFlt = original spatial filter
  switch
    Spatial:
      set MMPredicate // <Fld>, <AFlt>, <MMSFlt>
      set DKPredicate // <Fld>, <AFlt>, <DKSFlt>
    ASpatial:
      set MMPredicate // <Fld>, <AFlt>, <MMSFlt>
      set DKPredicate // <Fld>, <AFlt>, <DKSFlt>
    ASpatial_Spaial:
      set MMPredicate // <Fld>, <MMAFlt>, <MMSFlt>
      set DKPredicate // <Fld>, <DKAFlt>, <DKSFlt>
    Field:
      set MMPredicate // <MMFld>, <AFlt>, <SFlt>
      set DKPredicate // <DKFld>, <AFlt>, <SFlt>
    Field_Spatial:
      set MMPredicate // <MMFld>, <AFlt>, <MMSFlt>
      set DKPredicate // <DKFld>, <AFlt>, <DKSFlt>
    Field_Aspatial:
      set MMPredicate // <MMFld>, <MMAFlt>, <SFlt>
      set DKPredicate // <DKFld>, <DKAFlt>, <SFlt>
    Field_Aspatial_Spatial:
      set MMPredicate // <MMFld>, <MMAFlt>, <MMSFlt>
      set DKPredicate // <DKFld>, <DKAFlt>, <DKSFlt>
  endswitch
  return {MMPredicate, DKPredicate}
end.
```

(그림 15) 프리디킷 매칭 알고리즘

5. 구현 및 평가

5.1 구현 환경

본 논문에서 질의 분류 컴포넌트를 구현하기 위하여 사용된 시스템 환경은 다음과 같다. 질의 분류 컴포넌트의 개발에 C++ 언어가 사용되었으며, 구동에 사용된 시스템은 Pentium XEON 1.8Ghz, RAM 512MB, HDD 20GB의 하드웨어 환경에 Red hat Linux 9.0 O/S를가지고 있다. DBMS는 Geomania Millenium Server/Cluster 시스템이 사용되었다. 이때, 클러스터 시스템을 구성하고 있는 각각의 노드는 메인메모리 스냅샷을 관리 할 수 있는 컴포넌트가 추가된 다중레벨 DBMS를 사용하고, 로컬 시스템에서 테스트를 수행하였다. 또한 시스템 관리를 위하여 관리자 도구를 구현하여 이용하였다.

5.2 수행 결과

질의 분류 컴포넌트가 구동되고 있는 시스템을 수행하기 위하여 다음과 같이 Celluar PhoneUser, Car, Region의 릴레이션을 기반으로 응용 환경을 구축하였다. 이용된 테이블의 스키마는 (그림 16)과 같다.

```
CelluarPhoneUser
        (id MOID, position Mpoint, phonenumber VARCHAR)

Car     (id MOID, position MPoint, type string)

Region (id MOID, area Mpolygon, name string)
```

(그림 16) 예제 테이블 스키마

수행에 이용된 데이터들의 지역은 서울시이며, 이동객체들은 핸드폰 사용자, 자동차로 정의하였다. 또한 수행을 위하여 다음과 같은 모의 환경을 설정하였다.

모든 데이터들은 디스크 데이터베이스에서 처리되는 가운데 주말 오후 가장 이동객체가 많은 강남지역을 스냅샷으로 생성하여 메모리 데이터베이스에 현재 데이터를 구축하여 놓았다. 따라서 강남지역의 핸드폰 사용자, 자동차 이동 객체들은 Hot 데이터로써 모두 메모리 데이터베이스에서 현재 데이터로 관리되며, 과거 데이터는 디스크 데이터베이스에서 관리된다. 스냅샷은 (그림 17)의 질의를 통해 생성되었다. 강남지역 이외의 서울 지역은 Cold 데이터로써 디스크 데이터베이스에서 처리된다.

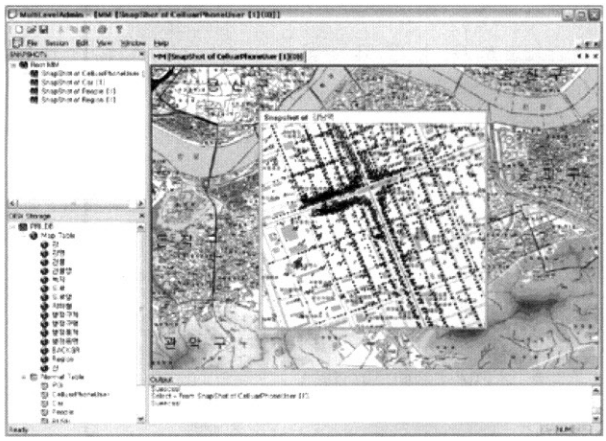
```
Select * From Region Where Region.name = '강남지역';

Select * From CelluarPhoneUser, Region Where
Contains(Extract(CelluarPhoneUser.position,NOW),Region.area)
= TRUE AND Region.name = '강남지역';

Select * From Car, Region Where
Contains(Extract(Car.position,NOW),Region.area)
= TRUE AND Region.name = '강남지역';
```

(그림 17) 스냅샷 생성 질의

서비스를 관리하기 위한 관리자 도구를 구현하여 수행하였다. (그림 18)은 관리자 도구의 실행 화면으로써 현재 스냅샷으로 생성되어 있는 강남지역의 이동객체인 핸드폰 사용자들을 나타내고 있다. 좌측 상단은 스냅샷 데이터를 보여주고 있으며, 좌측 하단은 디스크에서 관리되고 있는 데이터를 보여주고 있다. 질의 분류 컴포넌트를 구현함으로써 다음과 같은 환경에서 다중레벨의 데이터베이스를 사용하는 시스템의 변경만으로 사용자에게 빠른 서비스를 지원할 수 있게 되었다.

다음은 본 모의 환경에 기반을 둔 질의 분류를 설명하기 위하여 다음과 같이 세 개의 질의를 예를 들고, 각각의 질의를 분류하였다.

[질의 1] 현재 강남역 3번출구에 가장 가까운 핸드폰 사



(그림 18) 관리자 도구 실행 화면

용자 10명의 이름을 찾아라.

[질의 2] 현재 강남역 반경 100M와 공덕역 반경 100M에는 각각 몇 대의 자동차가 있는지 찾아라.

[질의 3] 어제 밤 11시 30분 구로역 1번 출구 반경 50M 이내의 핸드폰 번호를 모두 찾아라.

[질의 1]은 수행에 요구되는 데이터는 모두 스냅샷으로 생성되어 있기 때문에 메모리 질의로 분류되며, [질의 2]는 수행에 요구되는 데이터의 강남지역만이 스냅샷으로 생성되어 있으므로 나머지 지역은 디스크 데이터베이스에서 확인해야 하므로 하이브리드 질의이다. [질의 3]은 요구하는 지역과 데이터가 스냅샷으로 생성되어 있지 않고, 과거 데이터이므로, 디스크 질의로 분류할 수 있다.

5.3 성능 평가

5.3.1 실험 데이터

실험을 위하여 CitySimulator[15]에 의해 생성된 데이터를 이용하였다. CitySimulator는 3차원 위치를 모델링하여 최대 백만 명 까지의 이동 사용자들의 행동 패턴을 모의 실험할 수 있도록 가상의 데이터를 생성해주는 프로그램이다. 이 프로그램은 3차원 공간에서 사람들이 도로 상에서의 이동, 빌딩 간의 이동, 빌딩의 각 층간의 이동을 표현하는 등 실세계의 이동 환경을 모의실험 할 수 있도록 설계되었다. 이동 객체의 수를 각 50개, 200개, 400개, 800, 1000개로 지정하여, 객체가 이동하는 환경에서 분류된 질의의 종류에 따른 처리시간을 비교하였다.

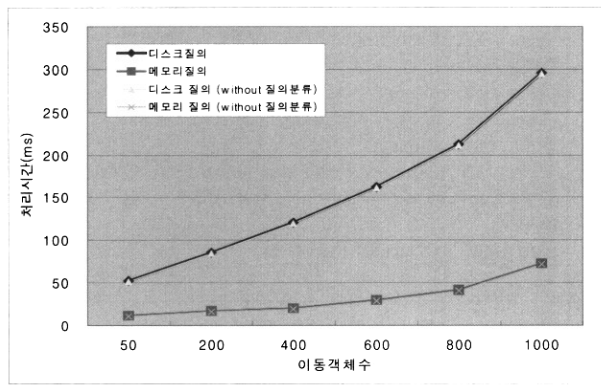
또한 하이브리드 질의의 경우 스냅샷으로 생성되어 있으면서 질의로부터 요구되는 데이터의 비율을 0%에서 100%까지 10%씩 변화시키면서 성능을 평가하였다. 실험에 사용된 질의는 "강남역 주변 반경 2Km에 있는 모든 핸드폰 사용자의 위치를 찾아라." 이다.

5.3.2 질의 성능 비교

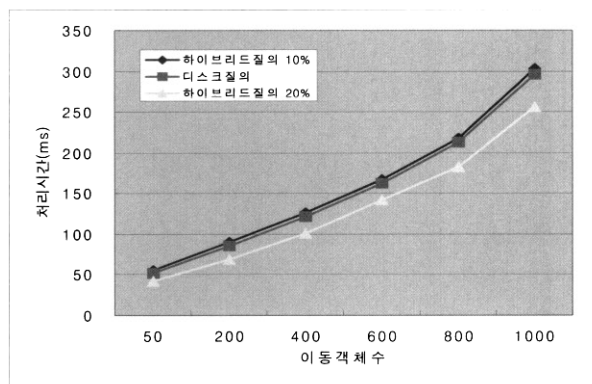
본 논문에서는 구현된 질의 분류 컴포넌트의 성능 평가를 위하여 동일한 질의를 사용하였으며, 요구되는 메모리 데이터의 영역을 변화시켜 가면서 테스트를 수행하였다. 따라서, 모든 데이터를 디스크에서 사용하는 디스크 질의와 모든 데이터를 메모리에서 사용하는 메모리 질의, 마지막으로 메모리와 디스크의 데이터를 공용하는 하이브리드 질의로 나누어 성능을 비교 하였다. (그림 19)는 디스크 질의와 메모리 질의를 비교한 그래프이다. 두개의 질의 모두 이동객체 수가 많아짐에 따라 처리 시간은 증가하지만 디스크 I/O에 대한 부하 때문에 디스크 질의의 처리 시간이 더욱 큰 폭으로 증가함을 알 수 있다. 그래프가 지수함수로 증가하는 이유는 업데이트 되고 있는 이동객체의 위치를 읽는데 있어서 락킹에 의한 지연이라고 볼 수 있다.

또한, 질의 분류 메커니즘을 적용한 질의와 질의 분류 메커니즘을 적용하지 않은 질의의 비교를 통해 각각 결과의 차이가 거의 없는 것을 알 수 있다. 따라서 질의분류 메커니즘은 디스크질의와 메모리질의에서 큰 부하가 아님을 알 수 있다.

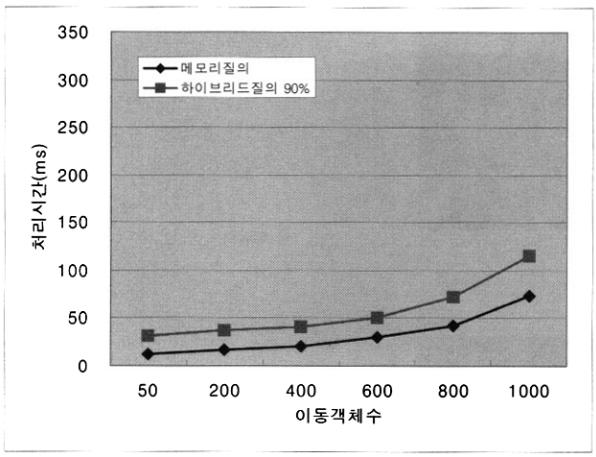
(그림 20)은 디스크 질의와 하이브리드 질의를 비교한 그래프이다. 하이브리드 질의는 각각 10%와 20%의 메모리 데이터를 이용하는 상황에서 테스트 하였다. 그 결과 10%의 하이브리드 질의는 디스크 질의의 처리시간보다 더 많은 시간이 걸린 것을 알 수 있다. 이것은 메모리에서 데이터를 읽는 시간의 이익보다 질의를 분류하는 메커니즘에 의한 부하로 판단할 수 있다. 또한 20% 하이브리드 질의는 디스크 질



(그림 19) 디스크 질의와 메모리 질의 비교



(그림 20) 디스크 질의와 하이브리드 질의 비교



(그림 21) 메모리 질의와 하이브리드 질의 비교

의 처리시간보다 더 적은 시간이 걸린 것을 알 수가 있다. 따라서 20% 미만의 구간에서는 선택적으로 디스크 질의로 분류하여 처리하는 것이 더 빠른 처리시간을 기대할 수 있다.

(그림 21)은 메모리와 하이브리드 질의를 비교한 그래프이다. 하이브리드 질의는 90%의 메모리 데이터를 이용한다. 그 결과 90%의 질의는 메모리 질의보다 예상대로 더 많은 시간이 걸린 것을 알 수가 있다.

질의간의 성능비교를 통해서 디스크 질의가 가장 많은 처리 시간이 걸리는 것을 알 수가 있으며, 다음으로 하이브리드 질의, 마지막으로 메모리 질의가 가장 적은 처리 시간이 소요되는 것을 알 수가 있었다. 그러나 하이브리드 질의는 대부분의 데이터를 디스크에서 사용하고 약 10%의 데이터를 메모리에서 사용할 경우, 질의 분류메커니즘의 부하로 인하여 오히려 디스크 질의에 비해 성능이 떨어지는 것을 알 수 있다. 따라서, 본 논문에서 사용한 시스템에서, 디스크 질의는 하이브리드질의 10%와 거의 같은 성능을 내는 것으로 판단되며, 이것은 하이브리드질의와 디스크 질의의 성능 임계점임을 알 수 있다.

6. 결 론

본 논문에서는 스냅샷을 갖는 다중레벨 DBMS에서 메모리 데이터베이스에 생성되어 있는 스냅샷을 최대한으로 이용하게 하는 질의 분류 컴포넌트를 설계 및 개발하였다. 질의 분류 컴포넌트는 입력된 질의를 메모리 질의, 디스크 질의, 하이브리드 질의의 세 가지로 분류하여 처리하였다. 특히, 하이브리드 질의의 경우 최대한으로 메모리 데이터를 사용하게 하기 위해 입력된 질의의 프리디킷을 메모리 부분과 디스크 부분으로 분할하여 처리함으로써 스냅샷의 사용율을 높였다.

제안된 질의분류 컴포넌트는 질의 분류 알고리즘을 적용하는 데에 따른 부하 때문에, 메모리 또는 디스크 질의의 경우, 질의 분류 없이 처리되는 경우보다 약간 더 많은 시간이 소요되는 단점이 있다. 그러나, 일반적인 LBS 환경에서 스냅샷을 갖는 다중레벨 시스템을 구축한 경우에는 주로 하이브리드 질의가 처리되기 때문에, 디스크와 메모리가 혼합하여 처리되는 특징으로 인하여 더욱 빠른 응답시간을 보장할 수 있다는 장점이 있다.

또한 질의별 성능 비교를 통해 메모리 질의가 디스크 질의에 비해 더욱 빠른 처리속도를 지원한다는 것을 확인하였으며, 본 논문에서 사용한 시스템에서 하이브리드 질의는 메모리 데이터의 사용율이 약 10%의 임계값을 넘는 경우에 디스크 질의보다 더욱 빠른 성능을 제공하는 것을 볼 수 있었다. 따라서 하이브리드 질의를 적절히 선택적으로 사용하는 것이 시스템 성능을 높일 수 있는 중요한 요소이다.

향후 연구로는 질의에 따른 성능 차이를 초점으로 한, 스냅샷을 갖는 다중레벨 DBMS에서 질의분류 메커니즘을 이용한 최적화 방법을 고려한다.

참 고 문 헌

[1] C. S. Jensen, A. Friis-Christensen, T. B. Pedersen, D. Pfoser, S. Saltenis and N. Tryfona, "Location-Based Services - A Database Perspective", Proceedings of the Eighth Scandinavian Research Conference on Geographical Information Science, As, pp.59-68, June, 2001.

[2] K. Virrantaus, J. Markkula, A. Garmash, V. Terziyan, J. Veijalainen, A. Katanosov and H. Tirri, "Developing GIS-supported Location-based Services", 2001. Proceedings of the Second International Conference on Web Information Systems Engineering, Vol.2, pp.66-75, Dec., 2001.

[3] D. H. Kim, K. H. Ryu and C. H. Park, "Design and Implementation of Spatiotemporal Database Query Processing system", Journal of Systems and Software, Vol. 60, Issue.1. pp.37-49, Jan., 2002.

[4] T. Sellis, "CHOROCHRONOS: Research on Spatiotemporal Database Systems", Proceedings of Tenth International Workshop on Database and Expert Systems Applications, pp.452-456, Sep., 1999.

[5] 신기수, 안윤애, 배종철, 정영진, 류근호, "GIS를 이용한 시공간 이동 객체 관리 시스템", 정보처리학회논문지D, Vol.8-D, No.2, pp.105-116, 2001.

[6] G. Trajcevski, O. Wolfson, B. Xu and P. Nelson, "Real-time Traffic Updates in Moving Objects Databases", Proceedings of 13th International Workshop on Database and Expert Systems Applications, pp.698-702, Sep., 2002.

[7] 유병섭, 이충호, 이재동, 배해영, "확장 가능한 고가용 데이터베이스에서 네트워크 비용을 줄이기 위한 변형된 분할기법", 한국정보과학회 춘계학술대회, Vol.29, No.1, pp.193-195, 2002.

[8] Michael Stonebraker, "Managing Persistent Objects in a Multi-Level Store", Proceedings of the ACM SIGMOD international conference on Management of data, pp.2-11,

1991.

[9] 장석규, 어상훈, 김명근, 배해영, "스냅샷 데이터를 갖는 다중 레벨 저장 DBMS에서 성능향상을 위한 질의 분류 방법", 데이터베이스 연구회 학술대회 Korean DataBase Conference, pp.121-126, 2005.

[10] C. Hsu and C. A. Knoblock, "Semantic Query Optimization for Query Plans of Heterogeneous Multidatabase Systems", IEEE Transactions on Knowledge and Data Engineering, Vol.12, Issue.6, pp.959-978, Nov/Dec., 2000.

[11] B. Harangsri, J. Shepherd and A. Ngu, "Query Classification in Multidatabase Systems", Proceedings of 7th Australasian Database Conference, pp.147-159, Jan., 1996.

[12] B. Harangsri, J. Shepherd and A. Ngu, "Query Optimisation in Multidatabase Systems using Query Classification", Proceedings of the ACM symposium on Applied Computing, pp.173-177, Feb., 1996.

[13] Zhu and P.A. Larson, "A Query Sampling Method for Estimating Local Cost Parameters in a Multidatabase System", In Data Engineering, pp.144-153, 1994.

[14] S. H. Eo, S. K. Jang, J. D. Lee and H. Y. Bae, "Multi-Level SDBMS with Snapshots", Proceedings of the 3rd ASGIS symposium, pp.283-294, Jun., 2005.

[15] http://www.alphaworks.ibm.com/tech/citysimulator. 2001.

장 석 규

e-mail : skjang@dblab.inha.ac.kr
2004년 인하대학교 컴퓨터공학부(공학사)
2004년~현재 인하대학교 대학원 컴퓨터·정보공학과(석사과정)
관심분야 : 다중레벨 데이터베이스, Mobile Computing, LBS/텔레메틱스, 유비쿼터스 컴퓨팅

어 상 훈

e-mail : eosanghun@dblab.inha.ac.kr
2003년 인하대학교 컴퓨터공학부(공학사)
2003년~현재 인하대학교 대학원 컴퓨터·정보공학과(통합과정)
관심분야 : 다중레벨 데이터베이스, LBS, 유비쿼터스 컴퓨팅, RFID 미들웨어

김 명 근

e-mail : kimmkeun@dblab.inha.ac.kr
1999년 인하대학교 전자계산공학과 (공학사)
2001년 인하대학교 대학원 전자계산공학과(공학석사)
2001년~현재 인하대학교 대학원 컴퓨터·정보공학과(박사과정)
터베이스 관리 시스템, 공간 데이터베이스

배 해 영

e-mail : hybae@inha.ac.kr
1974년 인하대학교 응용물리학과(공학사)
1978년 연세대학교 대학원 전자계산학과 (공학석사)
1989년 숭실대학교 대학원 전자계산학과 (공학박사)
1985년 Univ. of Houston 객원교수
1992년~1994년 인하대학교 전자계산소 소장
1982년~현재 인하대학교 컴퓨터공학부 교수
1999년~현재 지능형GIS연구센터 센터장
2000년~현재 중국 중경우전대학교 대학원 명예교수
2004년~현재 인하대학교 정보통신대학원 원장
관심분야 : 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스 등