# 클러스터 기반 분산 컴퓨팅 구조에서의 이동 객체 데이타베이스의 실시간 모니터링과 버퍼링 기법†

## Real-Time Monitoring and Buffering Strategy of Moving Object Databases on Cluster-based Distributed Computing Architecture

김상우* / Sang-Woo Kim　　이충우**** / Chung-Woo Lee
전세길** / Se-Gil Jeon　　황재일***** / Jae-Il Hwang
박승용*** / Seung-Yong Park　　나연묵****** / Yun-mook Nah

**요 약**

최근 무선 기술과 측위 기술의 발전에 따라 위치 기반 서비스에 대한 관심이 크게 증가하고 있다. 기존 연구의 단일 노드 기반 시스템으로는 처리하기 힘든 휴대폰 사용자와 같은 최소 백만 단위 이상의 대용량의 객체를 처리하기 위해 클러스터 기반 분산 컴퓨팅 구조로 GALIS가 제안되었다.

GALIS는 이동 객체의 현재 위치정보를 관리하는 SLDS와 과거 시간의 흐름에 따라 과거 위치정보를 관리하는 LLDS로 구성된다. SLDS와 LLDS는 분산된 다수의 노드로 구성되며 각 노드는 독립된 지역에 위치한 이동객체의 정보를 관리한다.

본 논문에서는 GALIS 프로토타입에 모니터링 기법을 적용하여 각 노드가 관리하는 이동 객체수의 변동에 따른 위치 정보관리 및 시공간 질의 처리 시 각 노드에 걸리는 부하를 모니터링 함으로써 각 노드의 역할을 동적으로 조정할 수 있게 각 노드별 이동 객체 처리상황 및 부하를 관리한다.

또, GALIS 질의 처리 서브시스템 성능 향상을 위해 질의 처리 결과를 버퍼링하여 연속된 질의처리 시 발생할 수 있는 중첩된 질의 영역을 관리하는 버퍼링 기법을 제안한다. 버퍼링 기법을 통해 수행되는 질의는 중첩된 질의 영역을 제외한 부분으로 나누어 수행하기 때문에 시스템으로부터 구해야 하는 결과 세트의 크기를 줄여주는 역할을 하여 질의 처리 수행시간이 감소하도록 질의 처리 과정을 개선한다.

**Abstract**

LBS (Location-Based Service) systems have become a serious subject for research and development since recent rapid advances in wireless communication technologies and position measurement technologies such as global positioning system. The architecture named the GALIS (Gracefully Aging Location Information System) has been suggested which is a cluster-based distributed computing system architecture to overcome performance losses and to efficiently handle a large volume of data, at least millions. The GALIS consists

† 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성·지원사업의 연구결과로 수행되었음.
■ 논문접수 : 2006.8.16　　■ 심사완료 : 2006.9.12
* 다믈 멀티미디어 연구원(swkim@dblab.dankook.ac.kr)
** 한국도로공사 도로교통기술원 책임 연구원(sgjeon@freeway.co.kr)
*** 단국대학교 전자 컴퓨터공학과 박사과정(sypark@dblab.dankook.ac.kr)
**** 알티베이스 연구원(cwlee@dblab.dankook.ac.kr)
***** 이비아이정보기술 연구소 소장(jihwang@dblab.dankook.ac.kr)
****** 교신저자 단국대학교 전자 컴퓨터 공학과 교수(ymnah@dku.edu)

of SLDS and LLDS. The SLDS manages current location information of moving objects and the LLDS manages past location information of moving objects.

In this thesis, we implement a monitoring technique for the GALIS prototype, to allow dynamic load balancing among multiple computing nodes by keeping track of the load of each node in real-time during the location data management and spatio-temporal query processing. We also propose a buffering technique which efficiently manages the query results having overlapped query regions to improve query processing performance of the GALIS. The proposed scheme reduces query processing time by eliminating unnecessary query execution on the overlapped regions with the previous queries.

**주요어 :** 이동 객체 데이터베이스, 위치 데이터 관리, 위치기반 서비스, GALIS

**Keyword :** Moving Object Database, Location Data Managements, Location-Based Service, GALIS

## 1. 서 론

최근 GPS(Global Positioning System)로 대표되는 측위 기술과 무선 통신 기술의 비약적인 발전, 그리고 이동 단말기의 대중화로 인하여 위치 기반 서비스(LBS: Location Based Service)에 대한 관심이 크게 증가하고 있다. 이에 대한 많은 연구들이 수행되었지만, 대부분의 연구가 단일 노드를 대상으로 하여 휴대폰 사용자 위치 추적 같은 위치 정보 갱신이 빈번하면서 최소 백만 단위 이상의 대용량 이동 객체를 처리하기에는 어려움이 있었고 이를 위한 많은 연구들이 수행되어져 왔다[1,2,3].

이러한 문제를 해결하기 위하여 제안된 GALIS(Gracefully Aging Location Information System) 아키텍처는 클러스터 기반 분산 컴퓨팅 구조로 설계되어 각 지역별 데이타를 여러 노드에 저장함으로써 위치 정보의 저장과 갱신 및 질의에 대한 부하를 분산시켜 대용량의 데이타를 처리할 수 있으며, 타임 존(time zone)이라는 개념을 도입하여 오래 된 데이타일수록 큰 정밀도의 질의 결과를 요구하지 않는다는 개념을 반영하여 각 시간대 별로 다른 정밀도를 설정하여 과거 위치 정보를 필터링하여 저장 공간 활용의 효율성 증대와 검색 시간의 감소가 이루어지게 하였다.

또한 GALIS에서는 분산 처리 지원과 스트리밍 데이타와 유사하게 지속적으로 발생하는 이동 객체 위치 저장에 필요한 실시간 처리를 위하여 효율성과 신뢰성이 높은 실시간 객체 모델인 TMO(Time-triggered Message-triggered Object) 구조를 적용하였다[2, 3].

GALIS는 이동 객체의 현재 위치정보를 관리하는 SLDS(Short-term Location Data Subsystem)와 과거 시간의 흐름에 따라 과거 위치정보를 관리하는 LLDS (Long-term Location Data Subsystem)가 주요한 구성요소이고 이동객체 데이터를 보고하는 이동객체 시뮬레이터와 질의 처리기로 구성된다. 객체는 시간의 흐름에 따라 그에 따른 타임 존을 이동하게 되는데, 이때 LLDS가 각 타임 존의 객체 정보를 변경하는 작업을 타임 존 시프팅(shifting)이라 한다[4,5].

본 논문에서는 GALIS 프로토타입에 모니터링 기법을 적용하여 TMO를 이용하여 발생되는 네트워크 부하 문제를 해결하여

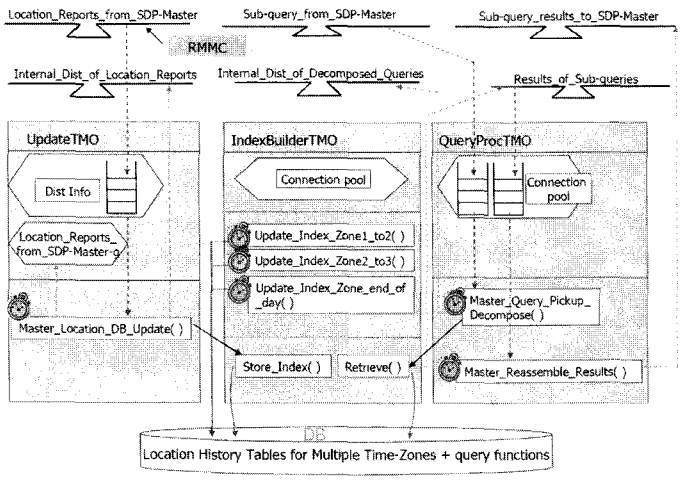
시GALIS 프로토타입의 성능을 향상시켰다. 또한 각 노드가 관리하는 이동 객체수의 변동에 따른 위치정보관리 및 시공간 질의 처리 시 각 노드에 걸리는 부하를 모니터링 함으로써 각 노드의 역할을 동적으로 조정할 수 있게 각 노드별 이동 객체 처리상황 및 부하를 관리하게 하였다. 또, GALIS 질의 처리 서브 시스템 성능 향상을 위해 질의 처리 결과를 버퍼링하여 연속된 질의 처리 시 발생할 수 있는 중첩된 질의 영역을 관리하는 버퍼링 기법을 제안하였다. 2장에서 관련 연구에 대해 소개하고, 3장과 4장에서 성능 평가를 위한 모니터링 시스템과 질의 처리 개선을 위한 버퍼링 기법을 각각 설명한다. 5장에서 시스템의 구현 및 버퍼링 기법의 성능평가에 대해 기술하고, 6장에서는 결론을 맺으며 향후 연구 과제를 제시한다.

## 2. 관련연구

### 2.1 TMO

본 논문에서 구현한 시스템은 객체 정보의 분산 처리와 이동 객체 저장의 실시간성을 보장하기 위해 실시간 객체 모델인 TMO(Time-triggered Message-Triggered Object)를 사용하였다. TMO 모델은 실시간 분산 처리의 요구를 충족하는 소프트웨어 컴포넌트이다[6]. 이전 방식의 분산 소프트웨어 컴포넌트에서 제공하는 전통적인 메시지 호출 방식의 Service Method(SvM)에 더하여 설계 단계에서 미리 명세된 시간 조건에 의해 구동되는(time triggered) Spontaneous Method(SpM)를 제공한다. TMO의 구조는 문법적으로 간단하지만 의미적으로 전통적인 객체 지향 디자인과 구현 기법의 확장이기 때문에 프로그래머는 보다 쉽게 접근할 수 있다[7]. TMO모델은 Sun Solaris나 Window NT 같은 상업적 플랫폼에 기반한 TMO 지원 설비 미들웨어가 생산되었고 이를 TMOSM(TMO Supported Middleware)이라고 한다. 이 미들웨어 아키텍처는 타이밍을 지원하는 실행의 정밀도 면에서 이전의 구현된 미들웨어 구조보다 더 효율적이다.

GALIS에서 독립된 프로세서에 의해 운영되는 SDP나 LDP는 3개의 TMO로 구성되어 있는데 각 TMO들은 TMOSM에서 제공하는 RMMC(Real-time Multicast and




<그림 1> LDP master 구조[10]

Memory replication Channel) 공유채널을 이용하여 통신한다.

3개의 TMO에는 각각의 역할에 따라 설계된 SpM이나 SvM을 가지고 있으며 그림 1은 LDP master의 한 노드에서 사용되는 TMO와 통신채널들을 보이고 있다.

### 2.2 GALIS 분산 대용량 이동 객체 관리 시스템 구조

GALIS의 전체적인 구조는 그림 2와 같다[8]. Moving Item Registry는 시스템이 관리하는 영역에 새로운 이동 객체가 출현하였을 때 Main DB에 이동 객체의 프로필을 등록하게 된다. GALIS는 크게 객체의 현재 위치정보를 처리하는SLDS(Short-term Location Data Subsystem)와 과거 위치 정보를 처리하는 LLDS(Long-term Location Data Subsystem)의 두 개의 서브시스템으로 구성되어 있다[9,10].

#### 2.2.1 SLDS

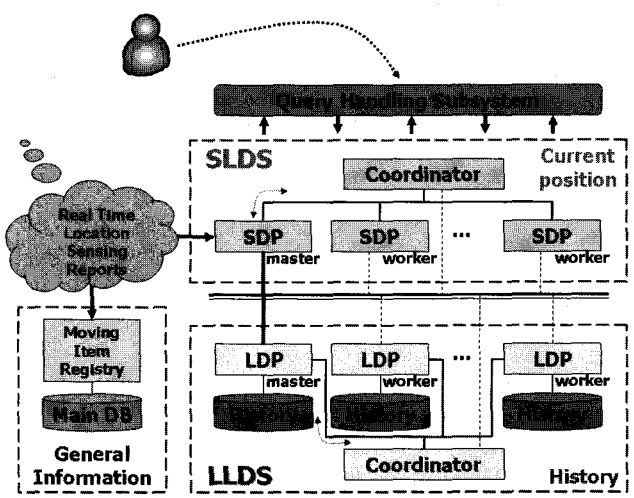
SLDS내의 각 노드를 SDP(Short-term Data Processor)라고 한다. SDP는 매크로 셀로 명명된 일정 지역 내의 객체의 정보를 담당하며, 다수의 SDP가 다른 객체들의 현재 위치 정보를 갱신하기 위해 동시에 동작하게 된다.

이동객체 데이타는 메인 메모리 데이타베이스를 사용하여 관리됨으로써 현재와 관련된 질의 수행 시 보다 빠르게 응답할 수 있고 빈번한 위치정보 갱신에 효과적으로 대응할 수 있다.

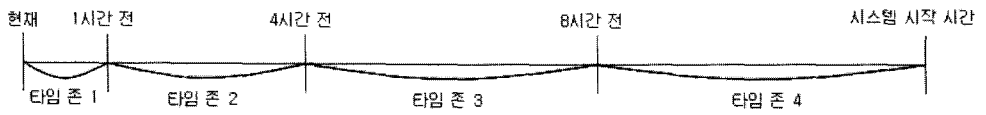
SDP master는 실시간으로 측위된 이동 객체의 위치 정보를 받아 다른 SDP worker로 전달하고, 새로운 위치정보가 들어오게 되면 과거 위치정보는 LDP(Long-term Data Processor) master로 전달하는 역할을 한다.

#### 2.2.2 LLDS

LLDS는 SLDS와 유사하게 하나의LDP master와 다수의 LDP worker들로 구성된다. LDP master는 SDP master로부터 이동객체의




<그림 2> GALIS의 전체 구조




<그림 3> 타임 존의 시간 영역

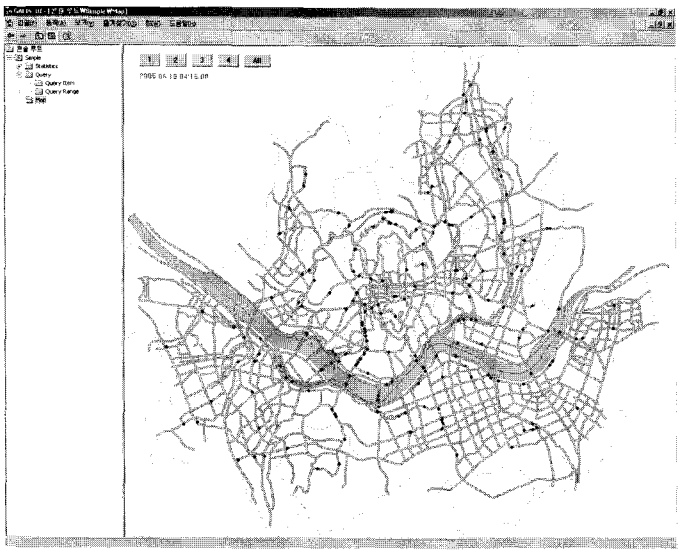
위치정보를 받아 다른 LDP worker로 전달한다. LLDS는 디스크 기반 데이타베이스를 사용함으로써 대용량의 데이터를 관리할 수 있다. 객체의 과거 위치 정보를 경과한 시간대에 따라 그림 3과 같이 네 개의 타임 존에 나누어 유지하게 된다. 일정 주기마다 자동으로 실행되는 메소드(Spontaneous Method)를 통해 시간의 흐름에 따라 데이타의 측위 시간을 기준으로 해당하는 타임 존으로 이동시키고, 각 타임 존마다의 기준 거리 이하의 이동 데이타를 필터링한다. 이 때, 타임 존의 수와 각 타임 존의 시간 범위는 응용 분야의 요구사항에 따라 설정할 수 있다.

## 3. 성능평가를 위한 모니터링 시스템

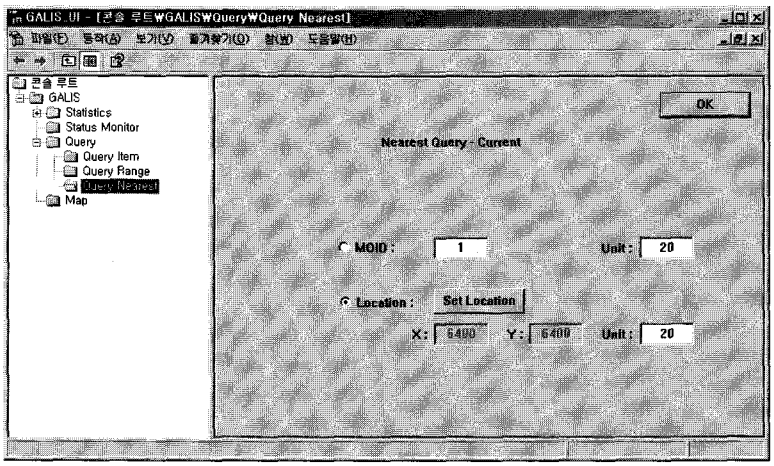
GALIS 모니터링 시스템은 이동객체 위치표시, 상황보고, 질의인터페이스 기능들을 MMC (Microsoft Management Console) 인터페이스에 맞춘 각각의 ActiveX Control 로 구성되어 다음과 같이 여섯 가지 사용자 인터페이스를 지원한다.

① Status Monitor SDP
② Status Monitor LDP
③ Query Item
④ Query Range
⑤ Query Nearest
⑥ Map

Status Monitor SDP와 Status Monitor LDP는 각각 SLDS와 LLDS에 속한 노드들의 상태 정보를 나타낸다. 질의 처리 인터페이스로 Query Item은 아이템 질의를 지원하고, Query Range는 영역 질의를 제공하며, Query Nearest 는 최 근접 질의를 할 수 있게 한다. 마지막으로 Map는 현재 보고되는 있는 이동객



<그림 4> 이동객체 위치표시




<그림 5> 최근접 질의 인터페이스

체의 위치를 표시한다.

### 3.1 이동객체 위치 표시

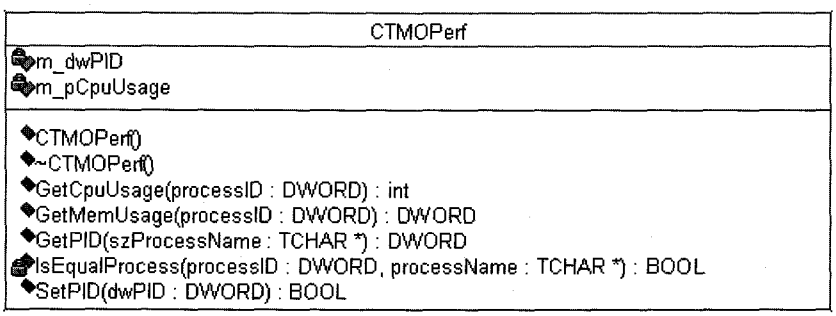
이동객체 시뮬레이터에서 보고된 위치정보는 SDP master를 통해 전달받아 GALIS 모니터로 보고한다. 그림 4와 같이 보고 받은 위치정보를 서울시 지도 위에 나타내어 직관적으로 이동 객체의 움직임을 식별할 수 있으며 관심 있는 영역을 확대해서 볼 수 있다.

### 3.2 질의 인터페이스

GALIS는 질의 인터페이스로 아이템 질의, 영역 질의, 최근접 질의를 지원한다. 그래픽 사용자 인터페이스를 통하여 관심 있는 영역을 설정할 수 있으며, 질의 처리 결과를 직관적으로 파악할 수 있다. 그림 5는 최근접 질의 인터페이스로서 이동 객체나 위치를 기반으로 가장 가까이 있는 이동객체의 수를 설정하여 위치정보를 알아볼 수 있다.

### 3.3 성능측정

각각의 TMO 노드들은 성능측정 라이브러리를 통해CPU 사용율, 메모리 사용량을 측정한 결과와 관리하는 객체 수를 GALIS 모니터에 보고함으로써 GALIS의 상태정보를 실시간으로 살펴




<그림 6> TMOPerf 클래스 다이어그램

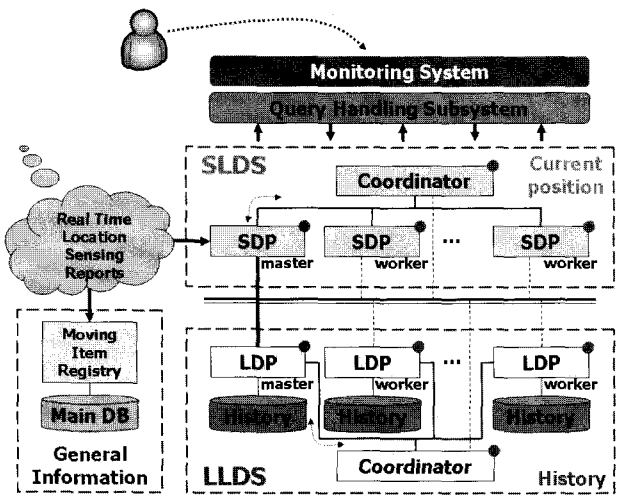
볼 수 있다. 그림 6은 GALIS 모니터링 시스템에 사용된 성능측정 라이브러리의 클래스 다이어그램이다.

TMO 노드는 TMOPerf를 사용하기 위해 자신의 프로세스 아이디(PID)값을 구해서 전달하거나 프로세스 이름을 전달함으로써 성능측정을 하고자 하는 대상을 설정해 둔다. 그리고 SpM 메소드와 같이 주기적인 처리가 수행되는 부분에서 TMOPerf를 통해 CPU 사용율, 메모리 사용량을 구한다. 각 노드는 현재 관리하고 있는 이동객체수와 함께 보고용 서버로 보고하게 된다. 추가적으로 노드의 유휴 CPU 사용율을 구할 수 있어 어느 노드가 추가적인 부하가 걸려도 괜찮은지 알 수 있다. 보고용 서버를 통해 전달받은 데이타를 통해 노드의 분할 및 합병을 결정하게 된다.

### 3.4 모니터링 정보 수집

질의 요청에 의해 다수의 노드에서 대용량의 이동객체 정보가 보고될 때와 실시간 상황보고가 수행될 때 실시간 분산 환경을 제공하는 TMO 의 통신채널인 RMMC 의 안정성을 위해 GALIS 상태 보고용 서버를 사용한다. 이는 RMMC를 이용한 모니터링을 위한 통신 채널을 줄여 주어, 기존 RMMC를 이용한 GALIS 시스템의 통신으로 발생되는 부하를 줄여주게 된다. 질의 종류 및 상황보고를 수행하는 노드마다 각각의 포트를 할당하였고 보고되는 가변적인 데이타 양에 유동적으로 대처할 수 있게 하여 기존 RMMC를 이용한 통신보다 부하를 줄여 실시간성을 높였다. 그림 7과 같이 각각의 노드는 성능측정 라이브러리를 사용하여 모니터링 시스템에 속해 있는 GALIS 보고용 서버로 직접 보고를 한다.

TMO는 통신채널에 부하가 걸리게 되면 전체 시스템에 영향을 미치게 된다. 이동객체의 저장 및 질의는 시스템 저하로 인한 실시간성의 보장을 받지 못하여도 크게 문제가 발생하지 않을 수 있다. 하지만 노드의 현재 상태를 보고하는 모니터링 시스템은 실시간성을 보장 받아야만 한다. 이러한 실시간성을 보장받기 위하여 보고용 서버를 설계하였다. 보고용 서버는 TCP/IP 소켓으로 구성되어 있으며 여러 TMO 노드들이 보고하




<그림 7> GALIS 모니터링 시스템 아키텍처

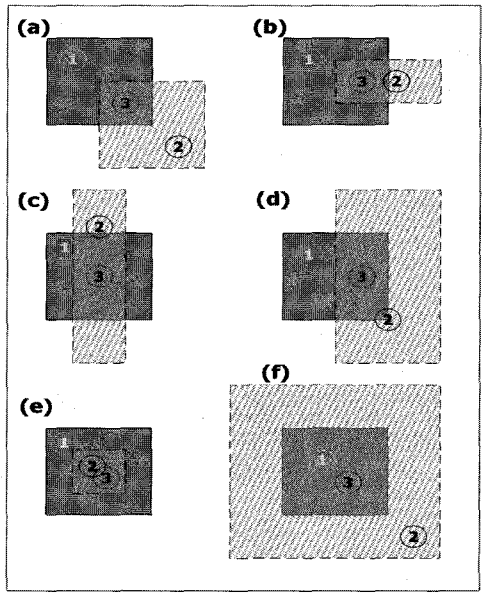
는 상황 보고 데이타를 처리한다. 또한 질의 처리 결과 데이타들도 보고용 서버를 통해 각 노드로부터 직접 수신한다. 보고용 서버를 사용함으로써 TMO 통신채널에 발생되는 추가적인 통신 비용 부담을 덜어준다.

## 4. 질의 처리 개선을 위한 버퍼링 기법

이동객체 정보를 관리하는 시스템에는 관심있는 영역에 집중된 질의 요청이 지속적으로 이루어진다. 본 논문에서 제안한 버퍼링 기법은 이동객체의 정보를 관리하는 시스템에 여러 개의 영역 질의가 수행될 때 발생될 수 있는 질의 영역간 중첩되는 영역을 버퍼링하여 관리하는 기법이며, 이를 통해 질의 처리 성능을 향상 시킬 수 있다. 본 연구는 시간영역에 대한 범위는 고정해두고 공간영역에 관한 질의 범위는 사각형으로 단순화하였다.

### 4.1 질의유형

질의 영역을 사각형으로 단순화시켜 중첩되는 경우를 유형별로 살펴보면 그림 8과 같다. 이전 질의한 영역을 일정하게 하고 유형별로 6가지 경우를 나타낸 것으로 ①은 이전 질의한 영역을 나타내고 점선으로 나타낸 ②는 현재 질의하고자 하는 영역을 나타낸다. ①과 ②가 겹치는 영역 ③은 이전 질의영역과 현재 질의영역이 중첩되는 영역을 나타낸다. 유형(a)는 질의할 영역이 이전 질의한 영역과 25%가 중첩되고, 중첩되지 않은 새로운 질의 영역이 육각형으로 생긴다. 유형(b)는 새로운 질의 영역이 사각형으로 생긴다. 유형(c)는 두 개의 사각형으로 새로운 질의 영역이 나뉘고 유형(d)의 경우는 팔각형으로 새로운 질의 영역이 생긴다. 유형(e)는 질의 영역 전체가 이전 질의 영역과 중첩된다. 유형(f)에서는 중첩되지 않는 영역이 도넛형태가 된다.




<그림 8> 중첩된 질의 유형

### 4.2 버퍼링 기법

그림 8의 ①,② 같은 영역 질의가 요청된다고 할 때, 두 질의는 ③과 같이 중첩되는 부분이 발생하게 된다. 제안한 버퍼링 기법은 먼저 요청된 ①번 영역에 대한 질의를 먼저 수행하고, 결과를 메모리에 버퍼링 시킨다. 이후에 수행되는 ②와 같은 영역 질의가 수행될 때, 이전 질의 영역인 ①과 중첩되는 영역이 있는지 검사하고 중첩되는 영역의 비율이 버퍼링 기법의 수행 계수보다 큰지 판단한다. 두 조건에 만족할 경우 중첩되지 않는 영역을 시스템에 질의하고 중첩되는 영역에 대한 질의는 버퍼링된 메모리에서 구해서 질의 처리결과를 합한다. 마지막으로 다음 영역 질의를 위해 시스템으로부터 구한 결과를 버퍼링 해둔다(그림 9).

질의영역을 사각형으로 단순화시켜 두 질의 영역 간 겹치는 영역이 있는지 판별해 본다. 이전 질의한 영역에 현재 질의한 영역의 꼭지점이 포함된다면 두 질의는 버퍼링 기법을 적용할 수 있다.

```
procedureImprovedQuery(RANGE cQ)
// cQ : current Query
// pQ : previous Query
// cQ separate bQ(buffer) and iQ(improved)
// Cr : coefficient of overlapped region ratio
begin
if(Overlap(pQ,cQ) is true) then
   if(GetOverlappedRegion(cQ) > Cr) then
      SeparateQuery(cQ, bQ, iQ);
      resultImp = QueryToDB(iQ);
      resultBuf = QueryToBuf(bQ);
      resultSet = resultImp + resultBuf;
      StoreResultToBuffer(resultImp);
      return
   endif
endif
// curQry does not intersect with Buffer
resultSet = QueryToDB(cQ);
StroreResultToBuffer(resultSet);
end
```

<그림 9> 버퍼링 기법 의사코드

이와 같이 버퍼링 기법을 통해 수행되는 질의는 시스템에 전달할 질의 영역의 크기를 줄여주고, 반환해야 하는 질의 결과 셋의 크기를 감소시켜 데이타 접근 시간을 줄여주게 된다. 이러한 버퍼링 기법을 이용할 때 버퍼링 되는 질의의 수와 질의 결과의 양은 적용되는 GALIS 시스템이 적용되는 도메인에 따라 다르게 설정하여야 한다.

### 4.3 질의어 분해과정

일반적으로 그림 8. (a)의 ②에 대한 질의는 그림 10과 같이 수행되었으나, 제안한 버퍼링 기법은 그림 11과 같이 질의 영역을 분할하여 버퍼링된 메모리로부터 구할 수 있는 질의 영역(bQ)과 시스템으로부터 구할 수 있는 질의 영역(iQ)으로 분할한다. 그림 12는 영역 ②를 버퍼링 기법을 사용하여 중첩된 부분을 제외한 영역만을 시스템에 질의할 수 있게 개선된 질의어 예시이다.

```
SELECT pos_x, pos_y
FROM galis
WHERE    Contains(POLYGON((2464    3464,7464
3464,7464 8464,2464 8464)),OBJ);
```

<그림 10> 질의 영역 대한 일반적인 질의어 예시

그림 10과 같은 영역 질의가 수행될 때, 중첩된 영역을 파악하여 버퍼링된 이전 질의 처리결과와 현재 질의영역과 중첩되는 부분이 사각형 RANGE(2464 3464, 6000 3464, 6000 7000, 2464 7000)로 될 때, 버퍼링기법에 의해 그림 11과 같이 메모리에 저장된 결과로부터 구해야 하는 영역과 시스템에 질의해야 할 영역으로 나눈다.

```
procedure SeparateQuery(RANGE cQ, RANGE bQ,
RANGE iQ)
// seg_cQ : segment of current Query
// seg_pQ : segment of previous Query
// listPoint : list of point
begin
for (each segment of pQ) do
   for (each segment of cQ) do
      seg_pQ = GetSegmentFromRange(pQ);
      seg_cQ = GetSegmentFromRange(cQ);
      point = GetIntersectionPoint(seg_pQ, seg_cQ);
      if (point is validate) then
         Add point to listPoint;
      endif
   endfor
endfor
bQ = ChangePointsToRange(listPoint);
for (each vertex of cQ) then
   point = vertex;
   if(point is not contain from bQ) then
      Add point to listPoint;
   endif
endfor
iQ = ChangePointsToRange(listPoint);procedure
ImprovedQuery(RANGE cQ)
// cQ : current Query
// pQ : previous Query
// cQ separate bQ(buffer) and iQ(improved)
// Cr : coefficient of overlapped region ratio
begin
```

```
if(Overlap(pQ,cQ) is true) then
   if(GetOverlappedRegion(cQ) > Cr) then
      SeparateQuery(cQ, bQ, iQ);
      resultImp = QueryToDB(iQ);
      resultBuf = QueryToBuf(bQ);
      resultSet = resultImp + resultBuf;
      StoreResultToBuffer(resultImp);
      return
   endif
endif
// curQry does not intersect with Buffer
resultSet = QueryToDB(cQ);
StroreResultToBuffer(resultSet);
end
```

<그림 11> 질의 영역을 분할하는 의사코드

```
SELECT pos_x, pos_y
FROM galis
   WHERE Contains(POLYGON((6000 7000,6000
3464,7464 3464,7464 8464,2464 8464,2464 7000)),OBJ);
```

<그림 12> 버퍼링된 영역을 고려하여 수정된 질의어 예시

## 5. 구현 및 성능 평가

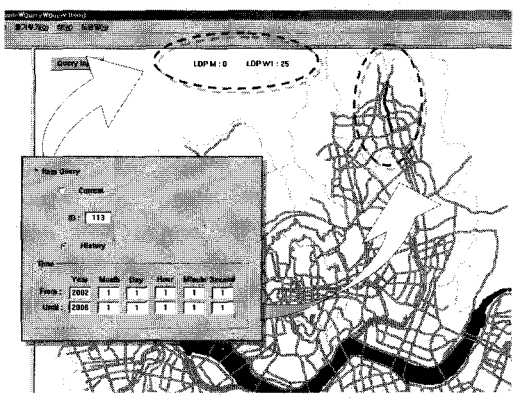
### 5.1 구현환경

개발된 프로토타입은 여섯 대의 컴퓨터로 구성하였다. 세 대의 컴퓨터는 SLDS 역할을 하며 메인 메모리 기반의 KAIROS 데이터베이스를 사용하여 지역 전체의 현재 이동 객체 데이타를 관리하는데, 그 중 두 대는 전체 지역을 2등분하여 각각 담당하고, 나머지 한 대는 유휴 노드로 분할, 합병 시나리오를 위해 준비한다. LLDS는 두 대의 PC로 구성되어 LDP master와 LDP worker로 동작한다. 두 노드는 전체 지역의 절반씩 담당하고 있으며, 위치정보를 저장하기 위하여 Geomania의 GMS 데이타베이스를 사용한다. GUI인터페이스를 통하여 질의를 입력 받고 결과를 출력하는 User Interface 및 모니터링 시스템이 나머지 한 대의 컴퓨터에 위치한다.

이동 객체가 움직이는 전체 영역의 범위는 40Km * 40Km로서 서울시의 벡터 맵을 사용하였다. 이동객체는 서울시 주요 도로를 따라 이동하게 생성하였다. 전체 영역은 2등분하여 SLDS는 모든 영역을 할당하고, LLDS는 전체영역을 각 PC에 1번 노드, 2번 노드를 할당하였다. 동시에 움직이는 이동객체의 수는 300개로 정하였고, 최초 300개의 임의의 좌표를 생성하여 매 30초 간격으로 객체들 움직임을 갱신하도록 설정하여 48시간 분량의 데이타 셋을 획득하여 실험에 사용하였다. 객체가 40Km * 40Km의 설정 영역 밖으로 이동하는 경우, 새 이동 경로가 영역의 경계선상의 교차점으로 강제로 이동시켜 객체가 영역을 이탈하는 경우는 없는 것으로 설정하였다.

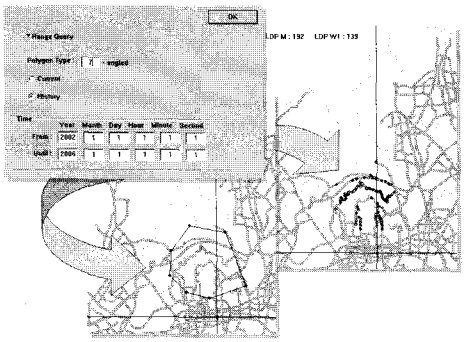
### 5.2 GALIS 프로토타입 실행 화면

그림 13은 본 논문에서 제안한 모니터링 시스템을 통하여 GALIS에서 아이템 질의 처리 과정을 보인 것으로 LLDS에 시간범위를 설정하고 이동객체 아이디를 113으로 설정하여 질의하여 LLDS의 각 노드로부터 질의 처리결과를 받아 화면상에 표시한 것이다. 각 노드들의 반환한 이동객체 정보를 간단히 화면상에 표시해 각 노드가 반환한 위치정보 결과를 살펴볼 수 있게 하였다.




<그림 13> 아이템 질의 처리 과정

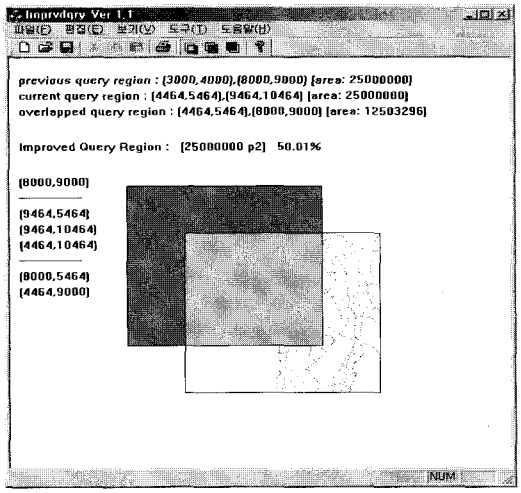
그림 14는 질의 영역을 나타내는 다각형 변의 수를 7로 하고 LDP master와 LDP worker1의 관리영역에 모두 해당하도록 범위 질의를 하여 그 결과를 이동 객체 아이디에 따라 각각의 색을 부여하여 나타내었다. 시간 조건(2002년 1월 1일 1시 1분 1초부터 2006년 1월 1일 1시 1분 1초까지)에 적합한 영역질의에 대해 LDP master와 LDP worker1이 각각 192건, 139건의 결과를 반환하였다.



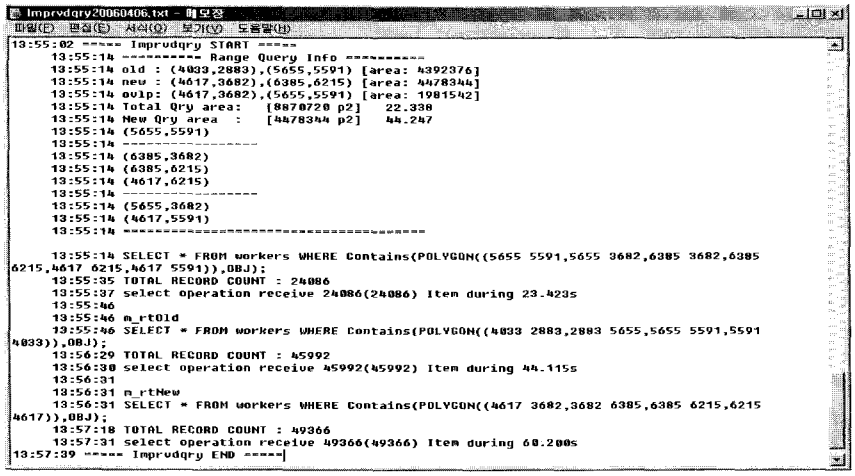
〈그림 14〉 영역 질의 처리 과정

## 5.3 버퍼링 기법의 성능 평가

버퍼링 기법의 성능 평가를 위하여 그림 15와 같이 중첩된 영역 질의가 수행될 때 버퍼링 기법의 성능 평가를 위한 프로그램을 작성하였다. 버퍼링된 이전 영역질의 결과와 현재 영역질의에 대한 정보를 나타내고 버퍼링 기법을 사용하여 중첩된 영역에 대한 정보를 나타내고 있다. 시스템에 질의해서 구해온 질의 결과, 버퍼링 기법에 의한 질의결과, 버퍼링된 이전 영역질의 결과를 각각 확인할 수 있게 하였다. 버퍼링 기법의 성능평가를 위해 적용한 영역질의는 사각형 질의만을 이용해 평가 하였다.



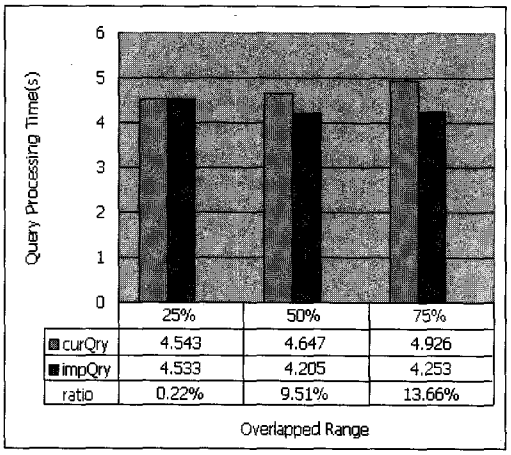

〈그림 15〉 버퍼링 기법을 활용한 영역 질의 실험화면




〈그림 16〉 버퍼링 기법 성능 평가를 측정한 결과 예시

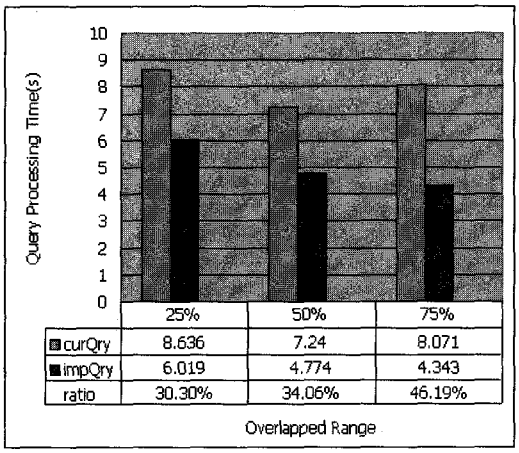
버퍼링 기법의 성능 평가를 위한 프로그램은 중첩된 영역 질의를 버퍼링 기법을 사용할 때와 그렇지 않을 경우를 각각 수행하면서 질의 처리 수행시간 및 개선된 질의 영역 범위, 영역 질의 처리결과의 수 등 성능 평가 항목을 그림 16과 같이 기록한다.

버퍼링 기법 성능 평가를 위해 이전 질의된 영역을 고정하여 버퍼링된 질의 결과들을 동일하게 하고 현재 질의 영역의 위치를 조정하여 중첩된 영역 비율을 25%, 50%, 75%로 높이면서 버퍼링 기법의 사용유무에 따른 질의 처리 수행에 걸리는 시간을 각각 측정하였다. 질의 영역의 크기는 1Km * 1Km, 2Km * 2Km, 4Km * 4Km, 8Km * 8Km의 조건으로 수행하여 결과를 그림 17, 18, 19, 20에 각각 나타내었다. 버퍼링 기법을 사용하지 않았을 경우(curQry)와 버퍼링 기법을 사용할 경우(impQry)에 따라 각각의 수행시간을 나타내고 있으며 버퍼링 기법을 사용할 경우 향상된 질의 처리시간 비율(ratio)을 중첩된 비율에 따라 나타내고 있다.



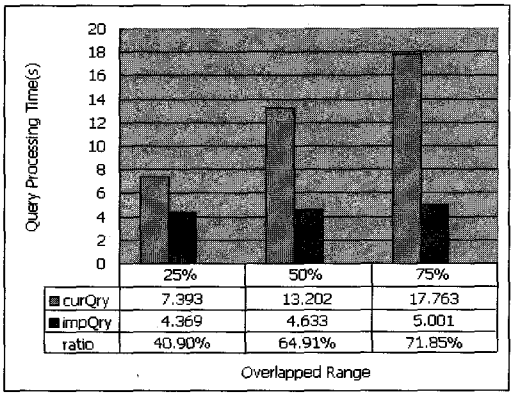

| | 25% | 50% | 75% |
|---|---|---|---|
| curQry | 4.543 | 4.647 | 4.926 |
| impQry | 4.533 | 4.205 | 4.253 |
| ratio | 0.22% | 9.51% | 13.66% |

<그림 17> 버퍼링 기법 성능 평가 (1Km * 1Km)



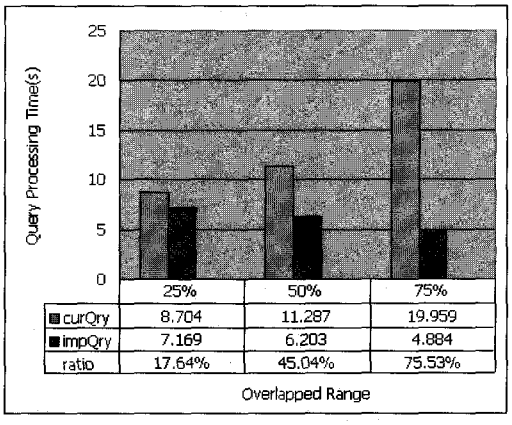

| | 25% | 50% | 75% |
|---|---|---|---|
| curQry | 8.636 | 7.24 | 8.071 |
| impQry | 6.019 | 4.774 | 4.343 |
| ratio | 30.30% | 34.06% | 46.19% |

<그림 18> 버퍼링 기법 성능 평가 (2Km * 2Km)




| | 25% | 50% | 75% |
|---|---|---|---|
| curQry | 7.393 | 13.202 | 17.763 |
| impQry | 4.369 | 4.633 | 5.001 |
| ratio | 40.90% | 64.91% | 71.85% |

<그림 19> 버퍼링 기법 성능 평가 (4Km * 4Km)




| | 25% | 50% | 75% |
|---|---|---|---|
| curQry | 8.704 | 11.287 | 19.959 |
| impQry | 7.169 | 6.203 | 4.884 |
| ratio | 17.64% | 45.04% | 75.53% |

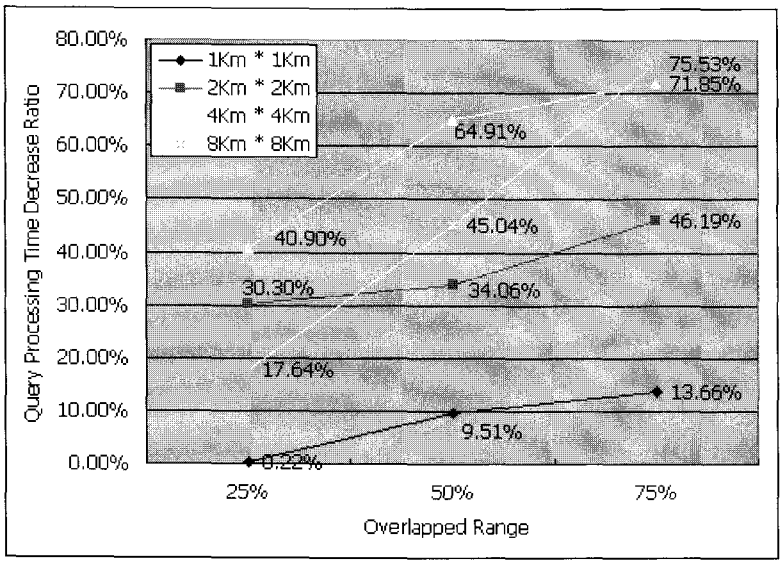
<그림 20> 버퍼링 기법 성능 평가 (8Km * 8Km)

결과처럼 중첩된 영역의 비율이 커질수록 버퍼링 기법을 활용한 질의 처리(impQry)는 버퍼링 기법을 사용하지 않은 질의 처리(curQry)에 비해 수행 시간이 점차 줄어드는 것을 알 수 있다. 이는 중첩된 영역이 커질수록 검색해야 하는 질의 영역이 작아지므로 디스크로부터 가지고 와야 하는 데이터의 수가 감소하기 때문에 나타나는 결과이다. 이전 질의한 영역을 동일하게 하고 질의 결과를 버퍼링한 후, 현재 질의하는 영역에 대한 처리시간이 다른 것은 이동객체 데이타의 분포가 동일하지 않기 때문에 중첩된 영역의 비율이 변화함에 따라 질의 영역이 이동하면서 질의 처리(curQry, impQry) 시간은 고정적이지 않다.

질의 영역을 1Km * 1Km로 설정한 그림 17의 경우 버퍼링 기법에 의한 성능 향상이 미세한 차이로 나타난다. 질의 영역 범위를 넓혀가면서 버퍼링 기법을 사용한 질의 처리에 필요한 수행시간과 그렇지 않은 기존 방식의 질의 처리에 필요한 수행시간에는 확연한 차이가 생기는 것을 알 수 있으며 설정한 영역마다 중첩된 비율이 증가할수록 버퍼링 기법에 의한 질의 처리 수행시간 감소가 이루어지는 것을 그림 21을 통해 알 수 있다.

## 6. 결 론

GALIS는 최소 백만 단위 이상의 대용량 이동 객체의 현재 위치 정보와 과거 위치정보를 관리하기 위해 설계된 시스템이다. GALIS 프로토타입 구현하던 중 발생한 통신 채널 부하에 의해 생기는 시스템 성능 저하 현상을 발견하였다. 이에 대한 해결책으로 성능 측정 라이브러리를 사용한 실시간 모니터링 기법을 도입하여 각 노드에 걸리는 부하를 실시간으로 관리하여 노드간 동적 부하 조정을 가능하게 하였다. 보고용 서버를 따로 둠으로써 질의 처리 결과를 TMO의 각 노드가 보고할 때 TMO 통신채널에 영향을 미치지 않게 하여 전체 시스템에 걸리는 부하를 최소화하였다. 또한, 연속된 질의 처리 시 중첩되는 영역에 대한 처




<그림 21> 중첩된 비율에 따른 버퍼링 기법의 성능

리를 버퍼링 기법을 도입하여 이전 질의 결과를 활용한 질의 처리의 성능을 개선하였고, 질의어를 분해하는 알고리즘을 제시하여 이를 시스템이 적용하였다. 버퍼링 기법은 연속적으로 영역 질의가 수행될 때 질의영역이 중첩되지 않은 부분만을 시스템에 질의하여 반환 받을 질의 처리 결과를 감소시키고 중첩된 부분을 버퍼링된 결과에서 구하여 질의 처리에 필요한 시간 비용을 대폭 감소시킨다. 본 논문에서 제시한 버퍼링 기법은 여러 가지 실험을 통해 그 효율성을 증명하였다.

모니터링 기법은 TMO의 통신 채널의 이용을 감소시켜 보다 전체 시스템 성능의 저하를 방지할 수 있다. 하지만 다른 통신 방법을 이용하여 추가적인 비용이 발생하는 단점이 있다. 또한 본 논문에서 제시한 버퍼링 기법은 사각형 질의만을 처리하기 때문에 원질의, 다각형 질의 등 다양한 영역 질의 유형에 대처하지 못하고 있다.

향후 연구 과제로는 다른 통신 채널을 이용하는 추가적인 비용 없이 GALIS 시스템의 성능을 향상시키는 연구와 본 논문에서 제안한 영역 질의의 효율적 처리를 위한 버퍼링 기법을 다양한 영역질의의 유형에 대처하는 연구 및 질의 처리결과를 큐에 삽입하여 처리하는 다단계 버퍼링 기법에 대한 연구가 있다.

## 참고문헌


1. T.K.Sellis, "Research Issues in Spatio-temporal Database Systems," Symposium on Spatial Databases Proc., pp.5-11, 1999.
2. IORT Inc., "Techniques for Storage and Indexing of the Very Large Volume of Moving Objects using Commercial Database Systems," Final Report to Electronics & Telecommunications Research Institute, December2003
3. J. Orenstein. Spatial query processing in an object-oriented database system. Proc. ACM SIGMOD, pages326-336, May 1986.
4. Moon Hae Kim, K.H.(Kane) Kim, Yunmook Nah, Joonwoo Lee, Taehyung Wang, Jonghoon Lee, Young Kyu Yang, "Distributed Adaptive Architecture for Managing Large Volumes of Moving Items," IDPT-Vol.2, 2003, pp.737-744.
5. Yunmook Nah, Moon Hae Kim, and Ki-Joon Han, "Distributed Scalable Approach for Managing Large Volumes of Location Data," in Proc. US-Korea Conference 2004, August 12-14, 2004, Research Triangle Park, North Carolina, USA.
6. Kim,K.H., "APIs for Real-Time Distributed Object Programming", IEEE Computer, June 2000, pp72-80.
7. Kane Kim, TMO Support Library(TMOSL) 3.0, UCI DREAM Lab, 2003.
8. Yunmook Nah, K.H.(Kane)Kim, Taehyung Wang, Moon Hae Kim, Jonghoon Lee, Young Kyu Yang, "GALIS: A Cluster-based Scalable Architecture for Location-Based Service Systems," Database Research, 18(4), KISS SIGDB, December 2002, pp.66-80.
9. Yunmook Nah, Moon Hae Kim, Taehyung Wang, K.H.(Kane)Kim and Young Kyu Yang, "TMO-structured Cluster-based Real-time Management of Location Data on Massive Volume of Moving Items," in Proc. *Workshop on Software Technologies for Future Embedded System* (WSTFES) 2003, IEEE Press, Hakodate, Japan, May2003, pp.89-92.

10. Yunmook Nah, K.H.(Kane) Kim, Taehyung Wang, Moon Hae Kim, Jonghoon Lee, Young Kyu Yang, "A Cluster-based TMO-structured Scalable Approach for Location Information Systems," in Proc. WORDS 2003 Fall, IEEE CS Press, October 2003, Capri Island, Italy.

**김상우**
2004년 단국대학교 컴퓨터공학과 공학사
2006년 단국대학교 전자컴퓨터공학과 공학석사
관심분야 : 이동객체 데이타베이스, 공간 데이타베이스, 분산 데이타베이스

**전세길**
1998년 단국대학교 컴퓨터공학과(공학사)
2000년 단국대학교 대학원 컴퓨터공학과(공학석사)
2004년 단국대학교 대학원 전자컴퓨터공학과(공학박사)
2006년~현재 한국도로공사 도로교통기술원
관심분야 : 데이타베이스, 멀티미디어, 시공간데이타베이스

**박승용**
1997년 단국대학교 컴퓨터 공학과 공학사
1999년 단국대학교 컴퓨터공학과 공학석사
1999년~현재 단국대학교 컴퓨터공학과 박사과정
관심분야 : 데이타 모델링, XML, 분산컴퓨팅, 이동객체 데이타베이스

**이충우**
1998년 단국대학교 컴퓨터공학과 공학사
2000년 단국대학교 컴퓨터공학과 공학석사
2000년~현재 단국대학교 컴퓨터공학과 박사과정
2002년~2005년 한국국방연구원 연구원
2005년~현재 알티베이스 선임연구원
관심분야 : LBS, 메인메모리 데이타베이스 시스템

**황재일**
1990년 단국대학교 전자공학과 공학사
1999년 단국대학교 산업대학원 정보처리학과 공학석사
2000년~현재 단국대학교 전자.컴퓨터공학과 박사과정
관심분야 : 데이타베이스, 멀티미디어, 시공간데이타베이스

**나연묵**
1986년 서울대학교 컴퓨터공학과 공학사.
1988년 서울대학교 대학원 컴퓨터공학과 공학석사.
1993년 서울대학교 대학원 컴퓨터공학과 공학박사.
1991년 IBM T. J. Watson 연구소 객원연구원.
2001년~2002년 University of California, Irvine 객원교수.
1993년~현재 단국대학교 전기전자컴퓨터공학부 전기전자컴퓨터공학 전공 부교수.
관심분야 : 데이타베이스, 데이타 모델링, 객체지향 데이타베이스, 멀티미디어 데이타베이스, 멀티미디어 정보 검색