# 분산 이동 객체 데이타베이스를 위한 과거 위치 정보 관리†

## Long-term Location Data Management for Distributed Moving Object Databases

이 호* / Ho Lee
이충우*** / Chung-Woo Lee
이준우** / Joon-Woo Lee
황재일**** / Jae-Il Hwang
박승용*** / Seung-Yong Park
나연묵***** / Yun-Mook Nah

**요 약**

최근의 위치 측위 기술과 무선 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 제시된 클러스터 기반 분산 컴퓨팅 구조로 GALIS가 제안되었다. GALIS는 이동 객체의 현재 위치정보를 관리하는 SLDS와 과거 시간의 흐름에 따라 과거 위치정보를 관리하는 LLDS로 구성된다. LLDS는 분산된 다수의 노드로 구성되며 각 노드는 독립된 지역에 위치한 이동객체의 정보를 관리한다.

본 논문에서는 이전의 GALIS 프로토타입에서 구현되지 않았던 이탈시간 관리 기법을 제안하여 노드간 이동 경로를 가진 이동객체를 추적하기 위한 질의유형에 대해 보다 정확하고 빠른 응답을 얻을 수 있음을 보인다.

LLDS는 객체의 과거 위치 정보가 타임 존을 이동할 때 필터링하여 저장하므로 보다 효율적인 저장 공간의 활용이 가능하다. 이때 LLDS가 모든 이동 객체의 위치 정보에 대해 해당 타임 존으로 이동시키고 정보를 필터링하는 작업을 타임 존 시프팅이라 한다. 본 논문에서는 GALIS에서 제안한 타임 존 시프팅을 구현하기 위해서 실시간 시프팅, 일괄 타임존 시프팅, 테이블 분할 시프팅 세 가지 기법을 제안하였고, 이를 구현하여 각 방법의 성능을 질의 테스트를 통해 제안된 세 가지 방법 중 테이블 분할 시프팅 방법이 보다 효율적임을 살펴볼 수 있었다.

**Abstract**

To handling the extreme situation that must manage positional information of a very large volume, at least millions of moving objects. A cluster-based scalable distributed computing system architecture, called the GALIS which consists of multiple data processors, each dedicated to keeping records relevant to a different geographical zone and a different time zone, was proposed.

In this paper, we proposed a valid time management and time-zone shifting scheme,

†본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성·지원사업의 연구결과로 수행되었음.
■ 논문접수 : 2006.8.16 ■ 심사완료 : 2006.9.12

* 엔티엘 아시아 연구원(hlee@dblab.dankook.ac.kr)
** 단국대학교 전자 컴퓨터공학과 박사과정(jwlee@dblab.dankook.ac.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)

which are essential in realizing the long-term location data subsystem of GALIS, but missed in our previous prototype development. We explain how to manage valid time of moving objects to avoid ambiguity of location information. We also describe time-zone shifting algorithm with three variations, such as Real Time - Time Zone Shifting, Batch - Time Zone Shifting, Table Partitioned Batch - Time Zone Shifting.
Through experiments related with query processing time and CPU utilization, we show the efficiency of the proposed time-zone shifting schemes.

**주요어 :** 이동 객체 데이터베이스, 위치 데이터 관리, 위치기반 서비스, GALIS
**Keyword :** Moving Object Database, Location Data Managements, Location-Based Service, GALIS

## 1. 서 론

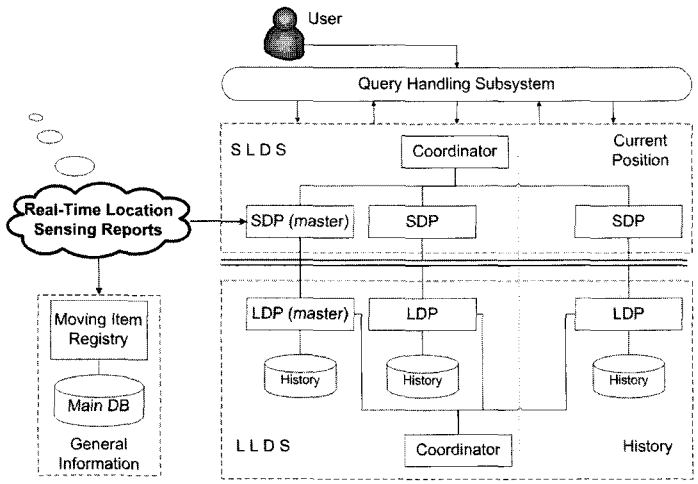
최근 GPS(Global Positioning System), RFID 나 센서 네트워크 등과 같은 위치 측위 기술과 무선 통신 기술의 비약적인 발전, 그리고 이동 단말기의 대중화로 인하여 위치 기반 서비스(LBS: Location Based Service)에 대한 관심이 크게 증가하고 있다. 대량의 위치정보를 관리하기 저장하고 질의하기 위해 RT-tree, 3D R-tree, 2+3 R-tree, STR-tree, TB-tree, hashing based index, TPR tree 와 같은 기법이 제안되었으나[1,2,3,4,5,6] 대부분의 연구가 단일 노드를 대상으로 하여 휴대폰 사용자 위치 추적 같은 위치 정보 갱신이 빈번하면서 최소 백만 단위 이상의 대용량 이동 객체를 처리하는데 어려움이 있었다.

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 있다[7]. GALIS에서는 타임 존(Time-zone)이라는 개념을 도입하여 오래된 데이타일수록 큰 정밀도의 질의 결과를 요구하지 않는다는 개념을 반영하여 각 시간대 별로 다른 정밀도를 설정하여 과거 위치 정보를 필터링하여 저장 공간 활용의 효율성 증대와 빠른 검색이 가능하도록 하였다[8,9].

본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음

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.

본 논문에서는 GALIS 프로토타입[10]중 이동객체의 시간의 흐름에 따른 위치정보를 저장하는 LLDS를 개선하였고, 타임 존 시프팅 시에 필터링을 통해 손실되는 노드 간 이동 객체의 위치 데이타의 빠른 추적을 하기위해 위해 GALIS의 기존 저장 구조를 개선하였다. 이를 통해 경로 추적 질의 시에 시스템내의 통신비용을 줄이면서 적절한 응답이 가능하게 되었다. 또, 타임 존 시프팅 방법으로 실시간(realtime) 시프팅, 일괄(batch) 시프팅, 테이블 분할을 이용한 일괄 시프팅을 각각 구현하여 각각의 성능을 비교하였다.




<그림 1> GALIS 아키텍처[7]

본 논문의 구성은 2장에서 관련 연구에 대해 소개하고, 3장에서는 LLDS의 저장 구조와 세 가지 타임 존 시프팅 방법에 대해 소개한다. 4장에서는 시스템의 구현과 성능 평가에 대해 기술하고, 5장에서는 결론과 향후 연구 과제를 제시한다.

## 2. 관련연구

### 2. 1 GALIS 아키텍처

클러스터 기반 분산 컴퓨팅 구조로 설계된 GALIS 아키텍처[7,8,9,10]의 전체적인 구조는 그림 1과 같다. Moving Item Registry는 시스템이 관리하는 영역에 새로운 이동 객체가 출현하였을 때 main DB에 이동 객체의 프로필을 등록하게 된다. GALIS는 크게 객체의 현재 위치 정보를 처리하는 SLDS(Short-term Location Data Subsystem)와 과거 위치 정보를 처리하는 LLDS(Long-term Location Data Subsystem)의 두 개의 서브시스템으로 구성되어 있다.

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. LLDS는 SLDS와 유사하게 하나의 LDP master와 다수의 LDP worker들로 구성된다. LDP master는 SDP master로부터 이동객체의 위치 정보를 받아 다른 LDP worker로 전달한다. LLDS는 디스크 기반 데이타베이스를 사용하여 객체의 과거 위치 정보를 경과된 시간대에 따라 네 개의 타임 존에 나누어 유지하게 된다.

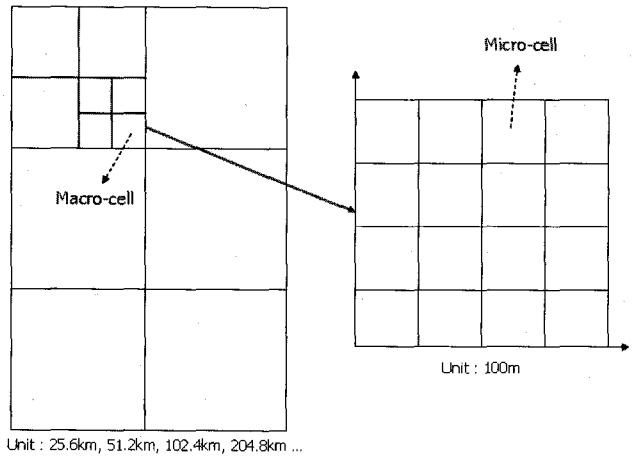
GALIS에서는 공간 영역에 그림 2와 같은 매크로 셀과 마이크로 셀로 불리는 2단계의

격자 구조(2-level grid structure)를 사용하고 있다[7]. 1단계의 매크로 셀은 한 변의 기본 길이 단위가 25.6Km이고 그의 배수로 늘어날 수 있는 정사각형의 셀이다. 하나의 매크로 셀에는 하나의 프로세서가 할당되어 셀 내의 이동 객체 정보를 관리하게 된다. 코디네이터가 주기적으로 각 노드의 상태를 파악하여 이동 객체가 집중된 노드를 분할하거나 이동 객체의 수가 적어진 노드들을 합병함으로써 각 매크로 셀들의 크기는 다를 수 있으며, 노드간의 동적 부하 조정(dynamic load balancing)이 가능하도록 한다.

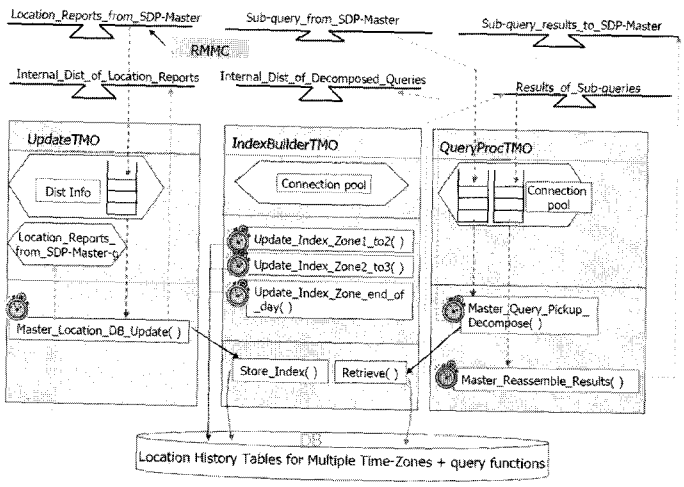
각 매크로 셀들은 마이크로 셀들로 분할된다. 2 단계의 마이크로 셀은 한 변의 기본 길이가 100m로 고정되어 균등한 크기를 갖는 정사각형의 셀이다. 균등 격자 구조를 지리적 영역에 적용함으로써 R-tree 계열의 인덱스를 적용했을 때 발생할 수 있는 인덱스 노드들의 잦은 분할과 합병을 피할 수 있는 이점이 있다[14]. 이렇게 동적 부하 균형을 제공하며 동시에 인덱스의 잦은 분할과 합병을 막아주는 2단계 격자 구조의 조합은 대형 위치 데이타 서버에 비용 효율적인 구조가 될 수 있다.

## 2.2 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[11]. 이전 방식의 분산 소프트웨어 컴포넌트에서 제공하는 전통적인 메시지 호출 방식의 Service Method(SvM)에 더하여 설계 단계에서 미리 명시된 시간 조건에 의해 구동되는(time triggered) Spon- taneous Method(SpM)를 제공한다. TMO의 구조는 문법적으로 간단하지만 의미적으로 전통적인 객체 지향 디자인과 구현 기법의 확장이기 때문에 프로그래머는 보다 쉽게 접근할 수 있다[12]. 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.




<그림 2> 2단계 격자 구조[7]




<그림 3> LDP master 구조[10]

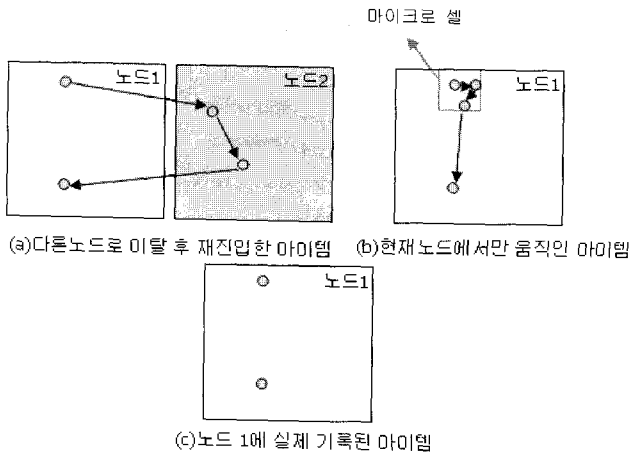
GALIS에서 독립된 프로세서에 의해 운영되는 SDP나 LDP는 3개의 TMO로 구성되어 있는데 각 TMO들은 TMOSM에서 제공하는 RMMC(Real-time Multicast and Memory replication Channel)[13] 공유채널을 이용하여 통신한다.

3개의 TMO에는 각자 역할에 따라 설계된 SpM이나 SvM을 가지고 있으며 그림3은 LDP master의 한 노드에서 사용되는 TMO와 통신채널들을 보이고 있다.

## 3. 이동객체 과거 위치정보 관리 기법

### 3.1 이탈 시간 관리

과거 위치정보를 관리하는 LLDS에서는 이동객체의 위치정보 레코드를 DB 에 저장하게 되는데 기존의 프로토타입에서는 위치정보에 포함되는 항목들은 OID(식별자), x_position, y_position(위치좌표), cell_ID(마




(a)다른노드로 이탈 후 재진입한 아이템 (b)현재노드에서만 움직인 아이템

(c)노드 1에 실제 기록된 아이템

<그림 4> 객체의 이동상화 및 기록 결과

이크로 셀 번호), time 이었다[10]. 그러나 이동객체의 현재 좌표에 위치한 시간을 time이라는 단일 항목으로 관리함으로서 일부 질의에 대해서 과다한 통신시간이 필요로 한다는 것을 발견하게 되었다.

그 예를 그림 4에서 나타내고 있는데, (a)는 이동객체의 움직임이 다른 노드로 이동하였다 다시 돌아온 상황을 표현한 것이고, (b)는 하나의 노드에서만 이동한 것을 표현한 것이다. 그러나 두 가지 다른 상황에서 이동객체의 움직임 정보는 노드1에 저장된 이동객체 자료는 (c)와 같이 기록된다. 이때 발생할 수 있는 문제는 (a)와 같이 다른 노드로 넘어갔다온 정보의 궤적을 알아내고자 하는 질의를 수행하려 하면 다른 노드에게도 질의를 수행하여 그 결과를 반환 받아야만 정확한 궤적을 추적할 수 있다. 이처럼 매크로 셀을 담당하는 각 노드는 다른 노드와 협업하지 않는 이상 이동 객체의 이동 경로를 정확히 파악할 수 없다. (c)와 같이 저장되어 관리되는 경우 이동 객체가 현재 노드를 벗어나서 다른 노드로 이동한 시간을 알 수 없으며 이는 그림 4.2와 같은 범위와 시간이 결합된 질의에 신뢰성 있는 결과를 반환하기 위해 많은 비용이 발생할 수 있는 요인이 된다. 이 문제를 해결하기 위해서, 위치 정보 의 time 항목을 start_time 과 end_time으로 구분하여 저장하였는데 start_time은 현재 레코드의 해당좌표에 진입한 시간을 의미하고 end_time 은 좌표에서 마지막 까지 위치했던 시간, 즉, 이탈시간을 의미한다.

LLDS에서 진입, 이탈시간을 저장 관리하기 위해서는 각각의 노드들은 자신이 관리하는 매크로 셀에 안에서 움직이는 이동객체뿐만 아니라 자신의 매크로 셀을 벗어나는 이동 객체에 대하여서도 추가적인 작업을 해야 한다. 즉, 자신의 노드에 위치했던 이동 객체가 다른 노드로 이동했는지를 판단하여, 자신의 노드에서 이탈한 시간을 기록해야 한다.

SLDS 로부터 위치 정보를 보고받는 LDP master는 보고 받은 위치 정보를 LDP worker들에게 브로드케스팅 하게 되는데, 만약 매크로 셀 단위의 이동이 있다면 RMMC를 통한 전송 정보(c_location_report)에 매크로 셀의 변화 여부 플래그(MMC_Flag)를 활성화 하여 LDP worker들

```
Set_MMC_flag(c_location report)
// MMC_flag (Macro Cell Change flag) :
//      set true when the moving object moves out of
//      the previous macro cell
// MO : a moving object
// c_location_report : current location report for MO
// p_location_report : previous location report for MO
begin
  for (each c_location_report) do
    If (c_location_report.macro_cell !=p_location_report.macro_cell)
    then
      Set c_location_report.MMC_flag is true;
    endif
  end for
end
```

<그림 5> Set_MMC_Flag 의사코드 (LDP master)

로 전송한다. 따라서 LDP master는 이전 주기 전체 이동 객체에 대한 매크로 셀 ID를 유지해야 하며 추가적인 메모리 자원(p_location_report)이 필요하게 된다(그림 5).

LDP worker 노드들은 LDP master로부터 RMMC를 통해 위치 정보를 수신할 때 자신의 매크로 셀 영역에 해당하는 이동 객체뿐만 아니라, 신규 보고된 이동 객체의 위치가 자신의 매크로 셀에 해당하지 않더라도 직전에 자신의 매크로 셀에 해당했다면 이탈 정보를 추가로 기록해야 한다.

노드에 진입한 이동 객체를 데이타베이스에 기록할 때 이탈 시간을 알 수 없으므로, 메모리 버퍼(PL_heap)상에 기록한다. 추후 노드를 이탈하는 시점에 데이타베이스에 기록하는 기법을 사용함으로서 하나의 이동 객체를 기록하기 위해서 중복되는 Disk IO를 감소시킬 수 있다(그림 6).

### 3.2 타임 존 시프팅

GALIS에서는 저장 공간 절약과 질의 성능

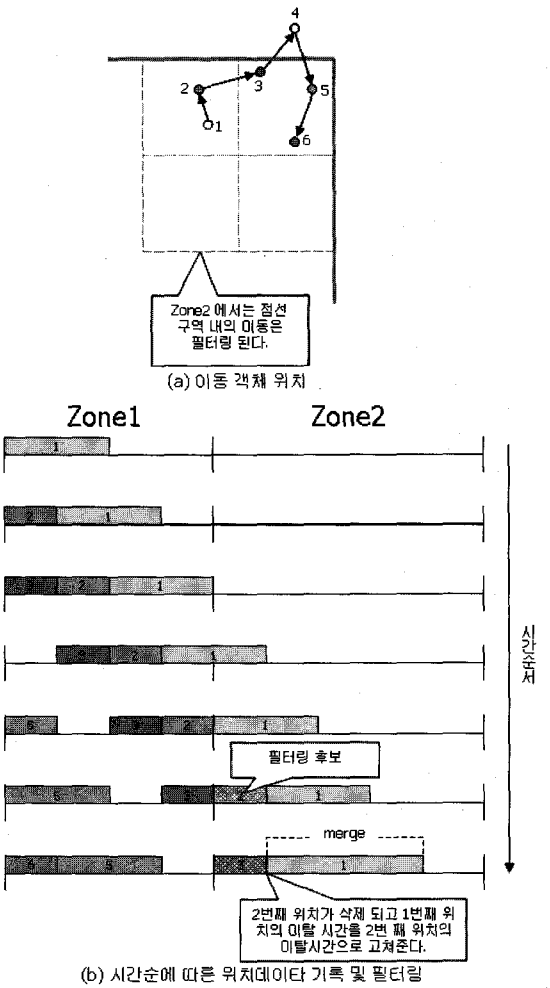
```
SvM_storeIndex(c_location_report, MCC_flag)
// MO: a moving object
// c_location_report : current location report for MO
// p_location_report : previous location report for MO
// MCC_flag (Macro Cell Change flag) :
//      set true when the moving object movies out of
//      the previous macro cell
// PL_heap : heap storing previous location reports temporarily
//           for moving objects belonging to the macro-cell
//           covered by the LDP node
begin
  if (MCC_flag is true) then
    // the macrocell of MO was changed
    if (OID is found in the PL_heap) then
      // MO left this node
      add end_time to the entry for MO in PL_heap;
      insert completed p_location_report to DB;
      remove the entry for MO from PL_heap;
    else
      // MO newly entered into this node
      add an entered into this node
    endif
  else
    // the macrocell of MO was not changed
    if (OID is found in the PL_heap) then
      // MO moved within this node
      add end_time to the entry for MO in PL_heap;
      insert completed p_location_report to DB;
      change the entry for MO in PL_heap;
    endif
  endif
end
```

<그림 6> SvM_storeIndex() 의사코드 (LDP worker)

향상을 위해 과거 데이타를 필터링 하는 타임 존이라는 개념[7]을 도입했다. '오래 된 데이타 일수록 세부적인 이동 정보를 필요로 하지 않는다.' 라는 개념을 반영하여 과거 데이타를 네 개의 타임 존으로 나누어 데이타베이스 상의 테이블로 구분하여 저장한다. 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 객체 데이타를 필터링한다. 이 때, 타임 존의 수와 각 타임 존의 시간 범위는 응용 분야의 요구사항에 따라 설정할 수 있다.

타임 존 필터링을 통한 데이타의 감소로 저장 공간 활용성을 높일 뿐만 아니라 데이타 필터링 및 시간대별 별도 테이블에 저장하여 질의 처리 시 질의 대상 테이블의 데이타를 적게 하여 빠른 질의 처리를 가능하게 하는 장점이 있다.

### 3.2.1 기본 알고리즘




<그림 7> 위치 데이타 필터링

그림 7은 특정 이동객체가 필터링 되는 상황에 대한 그림이다. (a)는 시간에 따른 이동객체의 위치 이동을 나타낸 그림이며 점선으로 표시된 영역 안에서의 이동은 존2에서는 필터링 함을 의미한다. 굵은 실선은 매크로 셀 영역을 의미하고 그림에서는 전체 매크로 셀 중에 오른쪽 상단만이 나타나 있다. (b) 그림은 시간에 따라서 실제 어떻게 기록되고 필터링 되는지를 시간순서대로 표현한 것이다.

그림은 SLDS로부터 총 6번의 위치가 보고되는 상황을 나타내고 있는데, 1번, 2번 위치 데이타는 존1 에서는 전부 기록되어 보존되고 있으나 시간이 흘러 신규 위치가 계속적으로 보고되어 존2 영역으로 이동하면 2번 위치는 점선으로 나타난 필터링 대상 영역 안에서만 이동하였으므로 필터링 되어 1번 기록과 합쳐진다. 마찬가지로 6번 위치도 필터링 되어 5번 위치 기록과 합쳐질 것이다.

필터링 후보로부터 필터링 대상을 가려내기 위해서는 이동객체의 이동범위를 계산해 내야 한다. 즉, 필터링 후보가 되는 시점의 바로 직전 위치를 알아야 하는데 현재 필터링 후보로 선정되었다면, 필터링이 끝난 후 버퍼영역(PSL_Buffer)에 저장시켜 두었다가

```
SpM_Update_Index()
// c_location_report : a candidate location report for MO in DB
// ps_location_report : previously shifted location report
//                      for MO in DB
// PSL_buffer : array storing previously shifted location
//              reports form source time-zone to target
//              time-zone temporarily for all moving objects
begin
  select all c_location_report whose end time equals the
    boundary time of this time zone from DB;
  for (each c_location_report) do
    if (OID is found in the PSL_buffer) then
      // object is already buffered during shifting
      if (c_location_report.start_time = PSL_buffer[OID].end_time
          & current location remains within the same medium cell)
      then
        // previous end time meets with start time update
        //   the end_time of ps_location_report
        //   for MO in DB;
        delete c_location_report for MO from DB;
        // candidate location report is filtered out
      endif
      change the cell_ID for MO in PSL_buffer;
    else
      // first occurrence of this object during shifting
      add c_location_report to PSL_buffer;
    endif
  end for
end
```

<그림 8> SpM_Update_Index() 의사코드

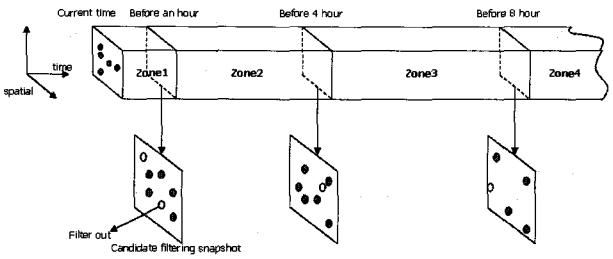
다음번 필터링 위치 정보와 비교하면 Disk IO를 줄일 수 있다. 필터링 대상으로 판명되어 이전의 위치 기록과 합칠 때에는 레코드 하나를 삭제하게 되는데 이전 위치 데이타의 이탈시간을 삭제할 위치 데이타의 이탈시간으로 고쳐준다(그림 8).

3.2.2 타임 존 시프팅 종류

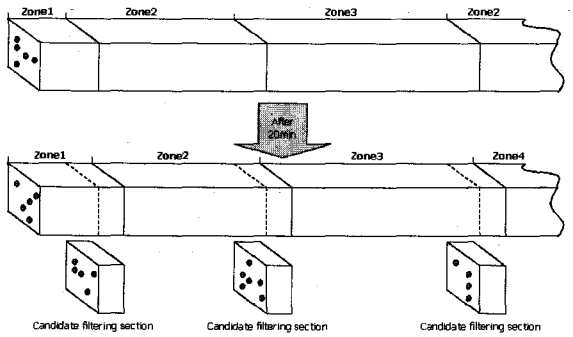
과거 위치정보를 필터링하는 SpM_Update_Index 메소드를 호출하는 시기에 따라서 다른 효과를 기대할 수 있는 SLDS 로부터 보고 받는 즉시 필터링을 수행 하는 것을 실시간 타임 존 시프팅(그림 9(a)) 이라고 한다.

만약 SLDS부터 매 30초 마다 위치 정보를 수신하게 된다면 LLDS 는 매 30초마다 필터링을 수행하게 된다.

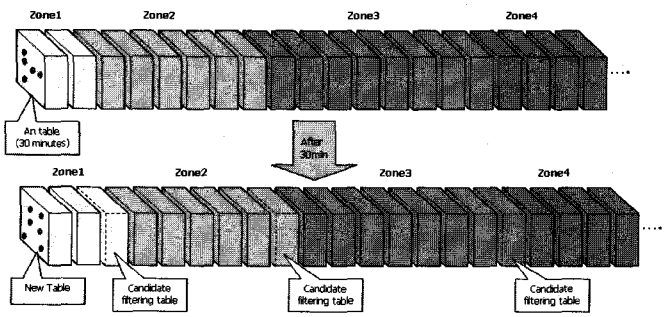
보고 주기마다 필터링을 수행하지 않고




(a) 실시간 타임 존 시프팅




(b) 일괄 타임 존 시프팅




(c) 테이블 분할 일괄 타임 존 시프팅

〈그림 9〉 타임 존 시프팅의 종류

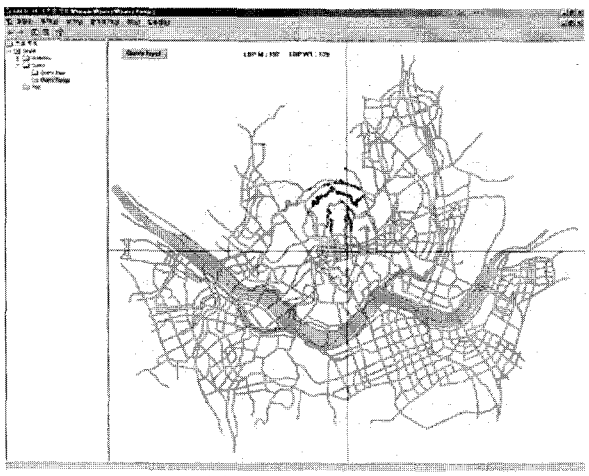
일정 시간 영역만큼을 지연했다가 한번에 수행하는 방식을 일괄 타임 존 시프팅 이라고 한다. 그림 9(b)는 매 20분 마다 해당 시간 영역에 대한 위치 정보를 일괄적으로 필터링 하는 예를 보이고 있다. 일괄 타임 존 시프팅은 보고 주기 당시의 시스템 부하를 줄일 수 있으나 일괄 처리 시점에서 많은 자원을 사용한다는 단점이 있다.

실시간 타임 존 시프팅 이나 일괄 타임 존 시프팅은 모두 이동 객체의 위치 정보를 단일 테이블 상에 저장 하는데 이것은 이동 객체가 늘어남에 따라 인덱스 갱신 비용이 증가하는 단점을 가진다. 이러한 단점을 극복하기 위해 위치 정보를 일정한 시간 별로 끊어서 서로 다른 테이블에 저장하고 이 테이블 단위로 필터링을 수행하는 방식을 테이블 분할 일괄 타임 존 시프팅 이라고 하며 그림 9(c) 는 매 30분 단위로 테이블을 생성하여 저장하는 예를 보이고 있다. 이러한 방식을 사용하면 저장뿐만 아니라 좁은 시간영역에 집중하는 질의에 효율적으로 대응할 수 있다.

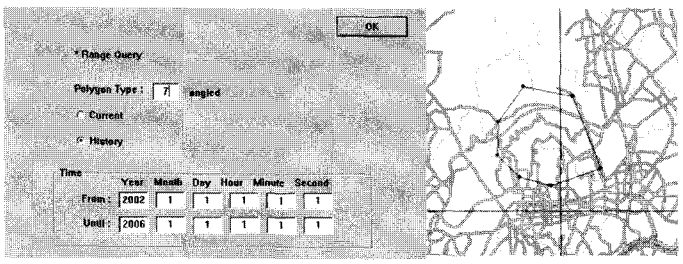
## 4. 실험

### 4.1 실험환경

실험을 위하여 구성된 테스트 시스템은 SDP 2개, LDP 2개, 총 4개의 노드로 이루어져 있으며 3대의 PC로 구성하였다. 3대의 PC는 모두 Pentium 4 2.6GHz CPU와

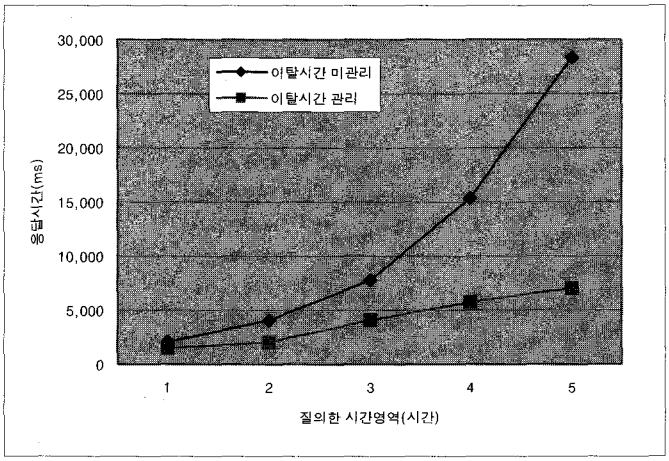


(a) 질의 응답 결과




(b) 질의 시간 입력 (c) 질의 영역 입력

<그림 10> 사용자 인터페이스




<그림 11> 이탈시간 관리에 따른 질의응답 시간

1Gbyte 의 RAM을 보유하고 있으며 Microsoft Windows 2003 Standard Server 를 OS로 사용한다.

PC1은 SLDS 역할을 하며 지역 전체의 위치 데이타를 관리한다. 또한 사용자 질의를 입력받고 결과를 출력하는 User Interface도 같은 컴퓨터에 위치한다. 다른 두 대의 PC는 LLDS 의 2개의 노드이며, LDP master, LDP worker로 동작한다. 두 노드는 전체지역의 절반씩 담당하고 있으며, 위치정보를 저장하기 위하여 Geomania의 GMS 데이타베이스를 설치하였다.

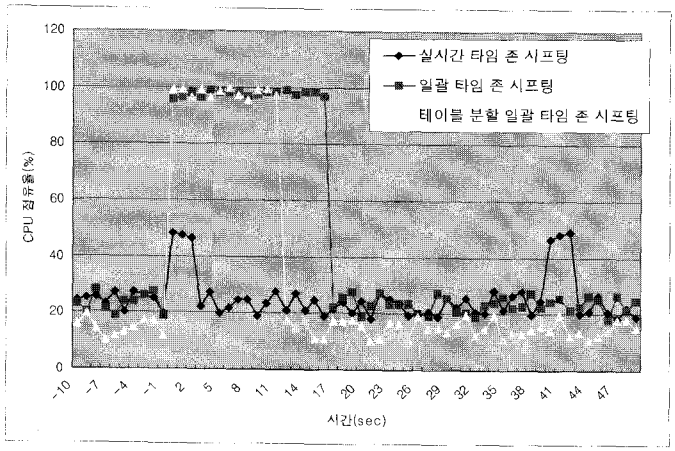
실험에 사용한 공간 영역은 40km x 40km 영역으로 서울시를 대상으로 하였고 모든 이동객체는 서울시의 주요 도로망을 따라 이동하도록 했다. 이동 객체 속도에 따라 보행자(5km/h), 저속 차량 탑승자(10km/h), 중속차량 탑승자(40km/h), 고속 차량 탑승자(80km/h) 로 구분하여 총 300개가 생성하였고 비율은 각 25%이다. 위치보고는 매 30초마다 주기적으로 한다. 그림 10은 테스트 시스템의 사용자 인터페이스로서 GALIS 시스템에 위치정보를 질의하는 화면이다.

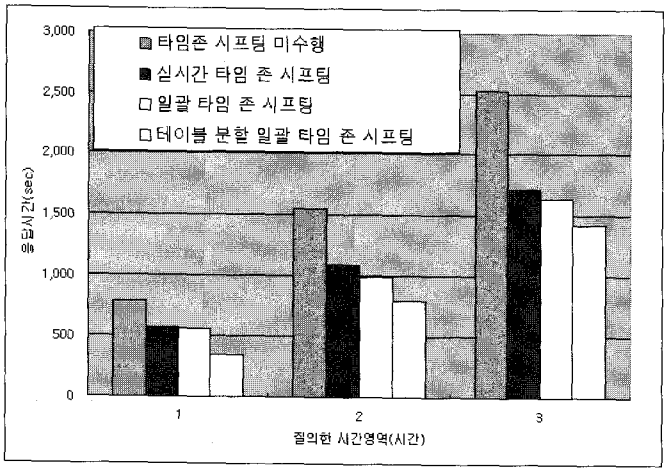
### 4.2 이탈시간 관리 전, 후 질의응답시간 측정

이탈시간 관리 기법을 적용하기 전, 후의 질의응답시간을 비교 평가하였다.

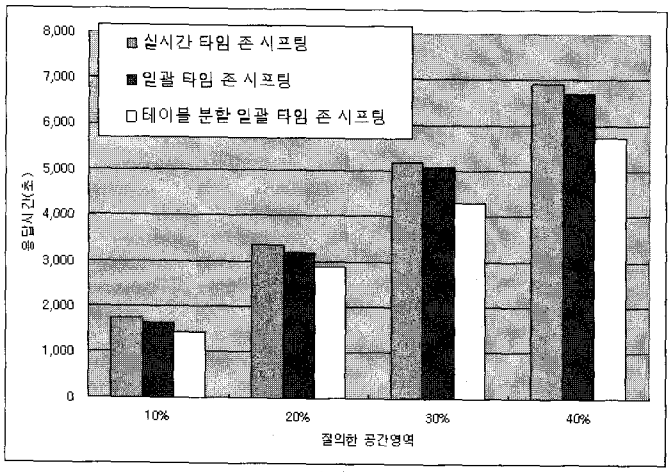
프로토타입에 24시간 분량의 데이타가 입력되어 있는 상태에서 노드1의 약 70% 영역에 대해서 5단계의 시간 영역에 대해서 질의 했다. 5단계의 시간 영역은 질의의 시간 조건을 시스템 시작부터 한 시간씩 증가시킨 것이다. 이탈 시간을 관리하지 않는 경우에는 질의 영역에서 이탈하거나, 새로 진입했을 경우를 알아내기 위하여 다른 노드와 추가로 통신해서 계산을 하고, 본 논문에서 제안한 이탈시간을 관리하는 경우에는 별도의 통신 시간이 없다.

그림 11은 5개 시간 영역에 대한 이탈 시간 관리 전후의 응답시간이다. 이탈시간을 관리하지 않는 경우에는 질의한 시간 영역이 늘어남에 따라 응답시간이 지수 적으로 늘어난다. 질의한 시간 영역이 늘어남에 따라 반환되는 이동객체의 수가 늘어나게 되는데, 진입, 이탈 여부를 판별하기 위해서 반환되는 이동객체 수만큼 다른 노드와 통신을 해야 하기 때문에 통신 시간이 추가되




(a)




(b)

〈그림 12〉 필터링 방법에 따른 질의응답 시간

어서 전체 반환시간이 증가하게 된다. 이탈 시간이 관리되는 경우에는 일정한 비율로 응답시간이 늘어남을 알 수 있다.

### 4.3 타임 존 시프팅 성능 비교

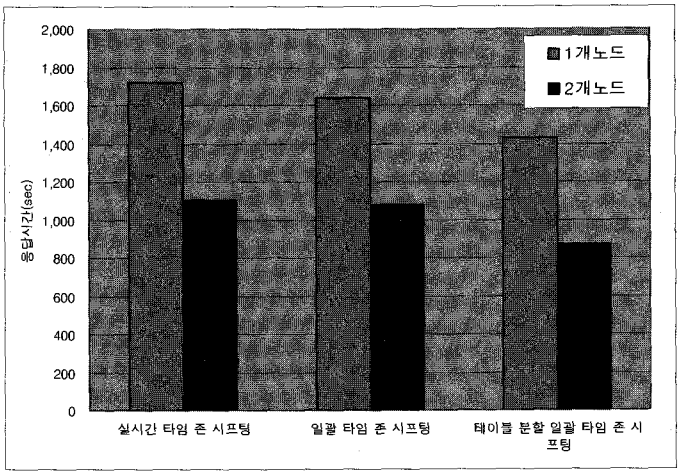
우선 필터링 방법에 따른 필터링 수행중의 CPU 사용율을 비교하였다. 타임 존 시프팅 연산이 일어나기 10초 전부터 1분간 초 단위로 측정하였으며 결과는 그림 12와 같다.

실시간 타임 존 시프팅은 매 보고 주기마다 필터링을 수행함으로 필터링 시의 CPU 사용률이 상대적으로 낮고 필터링 시간도 짧다. 일괄 타임 존 시프팅은 필터링시의 CPU 사용률이 100% 가까이 증가하고, 필터링 수행 시간은 약 16초 이다. 테이블 분할을 이용한 일괄 타임 존 시프팅 또한 CPU 사용률이 100% 가까이 증가하나 필터링 시간이 약 11초로 일괄 타임 존 시프팅보다 5초가량 감소되었다. 이것은 필터링에 따른 색인 갱신 비용이 감소된 결과로 볼 수 있다. 필터링 연산이 일어나지 않는 나머지 시간대, 즉 위치 데이타 삽입 연산만 일어나는 경우에는 테이블 분할을 이용한 타임 존 시프팅이 CPU 사용률이 가장 낮은데, 이것은 저장영역 분할에 따른 색인 갱신 감소의 결과로 볼 수 있다.

그림 12 에는 나타나지 않지만 실시간 타임 존 시프팅은 보고 주기인 매 30초 마다 필터링 연산을 수행하고, 일괄 타임 존 시프팅은 20분 간격으로 필터링 연산을 수행한다. 결과적으로 실시간 타임 존 시프팅은 일정하게 낮은 부하를 지속적으로 필터링 연산에 할당하고, 일괄 타임 존 시프팅은 순간적으로 높은 부하를 발생 시키나 빈도는 실시간 타임 존 시프팅보다 40배(30초 * 40주기 = 20분)가량 낮다.

그림 13는 이러한 CPU 사용이 일반적인 질의응답시간에 어떠한 영향을 미치는 가에 대해서는 실험 한 결과이다. 총 100개의 동일한 질의 셋을 테스트 시스템이 정상적으로 작동하고 있는 상황에서 연속적으로 입력 하여 마지막 100번째 결과가 완전히 반환될 때까지의 시간을 측정하였다. 따라서 평균 질의응답 시간은 그림의 결과에서 질의 갯수인 100으로 나누어야 한다.

그림 13(a)의 가로축은 시간 영역을 의미하는데 현재 시간부터 1시간, 2시간, 3시간 전까지의 이동객체의 위치를 추적하는 상황




<그림 13> 분산 환경에서의 질의응답 시간

이며, 공간 영역은 10%로 고정하였다. 그림 13(b)의 가로축은 공간 영역으로서 전체 영역의 10%, 20%, 30%, 40% 의 영역에 대한 질의로 나누어 실험하였으며 시간 영역은 현재부터 3시간 전까지로 고정하였다. 그림 13(a)에서는 필터링을 전혀 수행하지 않은 경우에 대한 결과도 포함하고 있는데 필터링을 수행함으로서 질의응답 시간이 대폭 감소됨을 볼 수 있다.

타임 존 시프팅을 통해 필터링을 수행한 데이타에 대한 질의응답시간은 일괄 타임 존 시프팅은 실시간 타임 존 시프팅과 비교하여 미세하게 좋은 성능을 나타낸다. 이것은 필터링 연산을 수행함에 있어 부하가 순간적으로 발생하지는 않지만 빈번한 필터링 연산이 사용자 질의응답에 대하여 좋지 않은 영향을 미친다는 사실을 알 수 있다.

주목할 만한 결과는 테이블 분할 일괄 타임 존 시프팅이 모든 경우의 질의에 있어 가장 좋은 성능을 나타낸 다는 것이다. 이것은 논리적으로 분리된 저장 공간에 위치 데이타를 저장함으로, 인덱스의 크기가 줄고 따라서 인덱스 갱신, 인덱스 탐색에 있어서 Disk IO가 줄어든 결과이다.

영역을 나누어 독립된 프로세서로 처리하는 개념은 GALIS의 기본 개념이며, 여러 실험을 통해 질의 처리 성능을 향상시킴을 입증 된 바 있다[9,10]. 타임 존 시프팅이 하나의 노드뿐만 아니라 다중 노드에 걸친 분산된 질의 처리 성능에 미치는 영향을 평가하기 위해 분산된 노드에 대해서 질의응답 시간을 측정해 보았다.

질의는 전체 시스템이 관리하는 영역의 10% 에 걸쳐 현재부터 3시간 전 까지의 데이타를 검색하는 형태이다. 이때 하나의 노드에 걸친 영역을 질의하는 경우와, 두 개의 노드에 걸친 영역을 질의하는 경우를 측정하였다. 그림 14는 그 결과이며 같은 영역 분산된 영역에 대한 질의 결과이다. 질의 처리를 분산처리 함으로 하나의 노드에 질의하는 경우보다 응답시간이 단축되었고, 앞서 실험 결과와 마찬가지로 3가지 타임 존 시프팅 중에서 테이블 분할을 이용한 일괄 타임 존 시프팅이 가장 좋은 성능을 나타내었다.

## 6. 결 론

GALIS는 최소 백만 단위 이상의 대용량 이동 객체의 현재 위치 정보와 과거 위치 정보를 관리하기 위해 설계된 시스템이다. 본 논문에서는 GALIS의 구조 중 과거 위치 정보를 관리하는 LLDS의 프로토타입을 TMO 아키텍처 기반과 공간 데이터베이스시스템인 GMS를 적용하여 구현하였고, 질의 수행을 보다 효율적으로 관리하기 위하여 저장 구조를 개선하였다.

이동궤적을 찾는 질의를 보다 효율적으로 수행할 수 있는 구조로 개선하기 위해 본 논문에서는 노드를 벗어나는 이동객체의 이탈 정보를 추가로 관리하는 구조를 제안하였다. 이탈 시간 관리를 통해 이동 객체의 궤적을 추적하는 질의에 대한 응답 성능은 노드간의 통신시간을 감소시키게 되어 이탈 시간 관리 전보다 질의응답 시간이 현저하게 줄게 되었다. 또한 이탈시간 관리의 효율적인 구조를 위해 메모리 버퍼를 이용하여 디스크 접근비용을 감소시켜 저장 성능을 향상시켰다.

또한 GALIS에서 제안한 타임 존 시프팅 개념을 적용시키고 타임 존 시프팅 시에는 대량의 데이타를 필터링하여 다른 테이블로 데이타를 이동시켜야 하기 때문에 이때 발생되는 노드의 부하는 지속적으로 발생하는 질의에 좋지 않은 영향을 미치게 된다. 이러한 부하를 줄이기 위해 본 논문에서는 실시간 시프팅, 일괄 시프팅, 테이블 분할 시프팅 이란 개념의 세 가지 방안을 제시 하였다. 본 논문에서 제안한 세 가지 방법 중 테

이블 분할 시프팅 방법이 가장 효율적임을 실험을 통해 살펴 볼 수 있는데, 이는 논리적으로 분리된 저장 공간에 위치 데이타를 저장함으로, 인덱스의 크기가 줄고 따라서 인덱스 갱신, 인덱스 탐색에 있어서 디스크 접근비용이 줄어든 결과이다.

이탈시간 관리기법은 메모리를 이용한 저장기법 개선으로 질의에 대한 응답속도를 높일 수는 있지만, 메모리를 사용하기 때문에 추가적인 저장 비용이 발생하는 단점이 있다. 또한 테이블 분할 시프팅 방법이 전반적인 질의 형태에 효율적일 수는 있지만 특정 질의는 분할된 테이블 전체를 검색 하여야 하기 때문에 질의 응답시간이 증가할 수도 있다. 향후 연구로는 이러한 단점을 개선할 수 있는 저장구조의 연구가 필요하다.

## 참고문헌


1. Xu, X., Han, J., and Lu, W., "RT-tree: An Improved R-tree Index Structure for Spatiotemporal Databases", *Proc. 4th Int'l Symp. on Spatial Data Handling (SDH), 1990.*
2. Theodoridis, Y, Y., Vazirgiannins, M., and Sellis, T., "Spatio-Temporal Indexing for Large Multimedia Applications", *Proc. 3rd IEEE Conf. on Multimedia Computing and Systems (ICMCS)*, 1996.
3. Nascimento, M.A., and Silva, J.R.O., "Towards Historical R-tree", *Proc. ACM Symp. on applied Computing (ACM-SAC)*, 1998
4. Pfoser, D. Jensen, C.S., and Theodoridis, Y., "Novel Approaches to the Indexing of Moving Object Trajectories," *Proc. Very Large Database(VLDB) 2000*, pp.395-406
5. Song, Z. and Roussopoulos, N., "Hashing Moving Objects," *Proc. Int'l Conf. on Mobile Data Management (MDM)*, 2001.
6. Saltenis, S., et al, Jensen, C., Leutenegger, S., Lopez, M., "Indexing the Positions of Continuously Moving Objects," *Proc. ACM SIGMOD*, 2000.
7. Nah, Y., Wang, T., Kim, K.H., Kim. M.H., and Yang, Y.K., "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, May 2003, pp.89-92.
8. Nah, Y., Kim, K.H., WangT., Kim, M.H., Lee, J., and Yang, Y.K. "A Cluster-based TMO-structured Scalable Approach for Location Infomation Systems," *in Proc. WORDS 2003 Fall*, IEEE CS Press, Capri Island, Italy, October 2003, pp.225-233
9. Nah, Y., Lee. J., Lee, W.J., Lee, H., Kim, M.H. and Han, K.J., "Distributed Scalable Location Data Management System based on the GALIS Architecture," *in Proc. Workshop on Object-Oriented Realtime Dependable Systems (WORDS) 2005*, February 2005, Sedona, Arizona, pp.397-404
10. Nah, Y., Lee, J., Park, S., Kim, S., Kim, M.H., and Han, K.J., "TMO-Structured Distributed Location Infomation System Prototype," *in Proc. International Symposium on Object and component-oriented Real-time distributed Computing (ISORC) 2005*, Seattle, Washington, May 2005, pp.321-328.
11. Kim, K.H., "Object Structures for

Real-Time Systems and Simulators", *IEEE Computer*, Aug. 1997, pp.62-70

12. Kim, K.H., "APIs for Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80
13. Kim, K.H., "Commanding and Reactive Control of Peripherals in the TMO Programming Scheme", *Proc. ISORC 2002 (5th IEEE CS Int'l Symp. on Object-Oriented Real-time distributed Computing)*, Crystal City, VA, April 2002, pp.448-456
14. Orenstein, J., "Spatial query processing in an object-oriented database system," *in Proc. ACM SIGMOD*, 1986.

**이 호**
2004년 단국대학교 컴퓨터공학과 공학사
2005년 단국대학교 컴퓨터공학과 공학석사
관심분야 : 이동객체 데이타베이스, 공간데이타베이스, XML

**이준우**
2000년 단국대학교 생물학과 이학사
2003년 단국대학교 대학원 컴퓨터공학과 공학석사
2003년~현재 단국대학교 대학원 컴퓨터공학과 박사과정
관심분야 : 이동객체 데이타베이스, GIS, 분산컴퓨팅, 바이오 인포매틱스

**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