GALIS의 디스크 기반 위치 정보 관리기의 설계 및 구현

고영균°, 나연묵
단국대학교 전자·컴퓨터 공학과
yggo@dankook.ac.kr, ymnah@dku.edu

Design and Implementation of Disk-based Location Information Manager of GALIS

Younggyun Go°, Yunmook Nah
Dept. of Electronics & Computer Engineering, Dankook University

요 약

최근 들어 이동통신 환경의 급격한 발달로 이를 활용한 위치기반 서비스에 대한 관심이 높아지고 있다. 효율적인 위치기반 서비스를 위해서는 실시간으로 위치를 변화시키는 이동객체에 대한 저장, 관리 및 질의를 담당할 수 있는 시공간 데이타베이스 관리 시스템의 존재가 필수적이다.

본 논문은 클러스터 기반 분산 컴퓨팅 구조를 바탕으로 제안된 시공간 데이타베이스 관리 시스템인 GALIS 구조 중에서 이동객체의 과거 위치 데이타를 디스크를 기반으로 저장 및 관리하는 노드인 LDP와 이동객체 데이터 생성기를 TMO 프로그래밍 스킴과 상용 데이타베이스 엔진을 사용하여 구현하였다. 제안 시스템은 대용량 이동객체의 효율적인 관리를 위한 실시간 엔진 개발에 활용될 수 있다.

1. 서 론

이동통신 기술의 발달과 이를 이용한 휴대전화, PDA 등의 무선기기의 보편화는 GPS와 같은 위치 측정 기술과 접목되어 다양한 종류의 위치기반 서비스(LBS: *Local Based Services*)가 가능하게 되었다.

위치기반 서비스의 경우, 통신기기의 사용자들로부터 GPS 수신기로 대량의 데이타들이 주기적으로 수신되며, 이들 데이타를 효율적으로 저장 관리를 할 수 있는 데이타베이스 시스템의 존재가 필수적이다. 이러한 데이터베이스 시스템을 시공간 데이타베이스 시스템이라 하며, 시공간 데이타베이스 시스템은 기존의 공간 데이타베이스 시스템과는 달리 지속적으로 위치가 변하는 이동객체의 현재 및 과거의 위치정보를 저장 관리해야한다. 또한 시공간 데이타베이스 시스템은 이동객체 데이터 모델의 정의와 이동객체를 위한 질의어 및 효율적인 인덱스구조가 필요하다. 시공간 데이터베이스 시스템에 대한 연구로는 이동객체의 질의어 구조에 대한 연구와 이동객체의 효율적인 저장 및 검색을 위한 인덱스 구조에 대한 연구가 진행되고 있으며, 이동객체의 현재 위치를 기반으로 과거위치 추적 및 미래위치 예측 연구가 진행되고 있다. 이동객체의 움직임에 대한 실제 데이터를 얻기 힘든 문제를 해결하기 위해 이동객체의 움직임 데이터 셋을 가상으로 생성시켜주는 데이터 생성기에 대한 연구가 진행되고 있다.

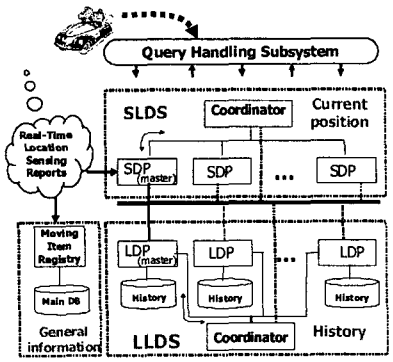
본 논문은 클러스터 기반 분산 컴퓨팅 구조의 시공간 데이타베이스 관리 시스템인 GALIS 구조 중에서 이동객체의 과거 데이타 부분을 관리하는 마스터 LDP(*Long-term Data Processor*)를 TMO 프로그래밍 스킴과 상용 데이터베이스엔진을 사용하여 구현한다[1,2,3].

본 논문의 구성은 다음과 같다. 2장에서는 실시간 이동객체 관리 시스템인 GALIS에 대하여 소개하고, 3장에서는 GALIS에 사용될 실시간 이동객체 데이터 생성기를 구현한다. 4장에서는 GALIS의 마스터 LDP의 실제 설계와 구현에 관하여 설명하며, 마지막으로 5장에서 결론 및 향후 연구 과제를 기술한다.

2. 실시간 이동객체 관리 시스템 GALIS의 개요

본 논문에서 사용한 GALIS(*Gracefully Aging Location Information System*)는 휴대전화 사용자와 같은 대량의 이동객체를 처리하기 위해 제안된 분산 컴퓨팅 구조이다. GALIS는 다중 데이터 프로세서로 구성된 클러스터-기반 분산 컴퓨팅 구조로서 각 프로세서가 특정 지역내에서의 이동객체의 움직임에 대한 처리을 담당한다. 특히, GALIS는 분산 컴퓨팅 환경을 위해 TMO 프로그래밍 스킴을 사용하여 시스템의 분석과 확장이 용이하며, 시스템의 저장 데이타베이스로 상용의 데이터베이스 엔진을 사용하기 때문에 비용 측면에 있어서도 매우 효율적이다. 또한 비균등 2-단계 그리드 구조를 사용하여 지리적 영역을 분할하기 때문에 이동객체가 특정 지역에 집중되는 상황에도 유연하게 대처할 수 있다. 이동객체의 과거 데이터를 저장하는 경우에도 TimeZone 이라는 시간 구역을 두어 시간대별로 그 위치의 정확도를 다르게 설정하여 결과적으로 데이터베이스의 낭비를 줄인다는 장점이 있다[1,2,3,4].

아래 [그림 1]은 GALIS 구조를 나타낸 것이다.

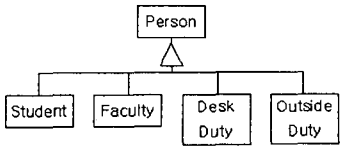


[그림 1] GALIS 구조

3. 실시간 이동객체 데이터 생성기의 설계 및 구현

본 실시간 이동객체 데이터 생성기는 GALIS의 성능 평가를 위하여 이동객체들의 움직임 데이터를 가상으로 산출하기 위한 목적으로 제작되었으며, 프로그램 상의 이동객체들은 고유한 움직임을 보여준다.

서로 다른 움직임 시나리오를 갖는 이동객체를 표현하기 위해서 본 프로그램은 이동객체를 5가지 종류로 분류하여 클래스화 하였으며, 이를 간단한 그림으로 표현하면 아래 [그림 2]와 같다.



[그림 2] 이동객체들의 클래스 상속도

[그림 2]에서 보여지는 바와 같이 자식 클래스인 Student, Faculty, DeskDuty, OutsideDuty는 부모 클래스인 Person의 상속을 받는다. Person클래스란 이동객체를 구성하는 데 필요한 부분만을 구현해 놓은 클래스로서 그 세부적인 구조는 아래 [그림 3]에 나타나 있다.

Person	
int Oid;	// 객체의 OID
POINT Cur_P;	// 객체의 현재 좌표
POINT Old_P;	// 객체의 이전 좌표
int TimeInterval;	// 경과 시간 값
int Cell_No;	// 객체의 소속 셀 번호
int Old_Cell_No;	// 객체의 이전 셀 번호
int TimeZone;	// GALIS의 TimeZone 값
int Reserve1;	// 예약값 1
int Reserve2;	// 예약값 2
int MovingMethod;	// 객체의 움직임 방법
int Direct;	// 객체의 움직임 방향
int Cal_Cell_No();	// 객체의 소속 셀번호 계산
bool obj_move();	// 객체의 움직임 결정

[그림 3] Person 클래스의 구조

Person클래스는 일정한 움직임 시나리오가 없이 단지 무작위적인 움직임만을 갖게 되는데, 이는 실제 현실에서 이동객체의 움직임을 나타내기엔 부적절한 것이다. 때문에 현실과 유사한 다양한 움직임 시나리오를 갖는 새로운 클래스들이 필요하게 된다.

본 데이터 생성기에서는 이러한 움직임을 갖는 이동객체들의 예로서 Student, Faculty, DeskDuty, OutsideDuty의 4가지 클래스를 정의하였으며, 위 클래스들은 부모클래스인 Person클래스를 상속받는다.

모든 자식클래스들은 유사한 부분이 많으므로 본 논문에서는 Student클래스를 예로서 설명하도록 하겠다. Student클래스는 학생이라는 이동객체의 움직임을 기초로 만들어진 클래스이다. 전술한 바와 같이 Student 클래스는 고유한 움직임 시나리오를 가지며, 아래 [그림 4]는 Student클래스의 움직임 시나리오를 나타낸 것이다.

[Student 클래스의 움직임 시나리오]
- 시작 좌표 (집) 및 목표 좌표 (학교) 설정
- S000 ~ S119 : 등교 (집 -> 학교로 이동)
- S120 ~ S359 : 수업 (50분 수업 후 10분 휴식)
- S360 ~ S480 : 하교 (학교 -> 집으로 이동)

[그림 4] Student클래스의 움직임 시나리오

Student클래스로부터 실체화된 모든 객체는 위 [그림 4]의 시나리오에 따라 움직이게 되는데, 그림에서의 S숫자 값이란 시스템의 시간간격(*Time Interval*)을 나타내는 것으로 GALIS의 시간간격은 30초이다.

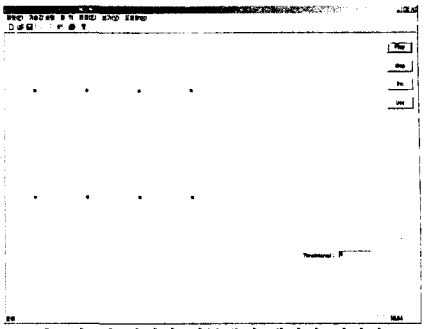
Student클래스의 기본 구조는 아래 [그림 5]와 같으며 부모클래스인 Person클래스에서 상속된 부분은 생략했다. [그림 5]에서 보여 지는 바와 같이, Student클래스에 새로이 추가된 부분들은 대부분 Student 클래스의 움직임 시나리오를 적용하기 위한 제어함수 및 변수이다.

Student		
RECT	SchlRect;	// 소속 학교의 좌표
RECT	HomeRect;	// 객체의 집 좌표
POINT	SchlCenter;	// 학교의 중심좌표
POINT	HomeCenter;	// 집의 중심좌표
int	Dist2Schl;	// 집과 학교의 거리
void Direct2Home(); // 학교로부터 집의 방향을 결정해주는 함수		
void Direct2Schl(); // 집으로부터 학교의 방향을 결정해주는 함수		
void Back2Home(); // 객체를 학교에서 집으로 움직이는 함수		
void StudyNRelax(); // 수업과 휴식시간의 움직임을 제어하는 함수		
void Home2Schl(); // 객체를 집에서 학교로 움직이는 함수		
void SelectSchoolPos(); // 해당 객체의 학교를 결정해주는 함수		

[그림 5] Student클래스의 구조

Student 클래스가 실체화되면, 우선 해당 객체의 집(*Home*) 좌표를 결정하고, 이후 지도상에서 가장 가까운 학교의 좌표를 찾아 자신의 학교로 지정한 후, 위 [그림 5]의 시나리오대로 움직이게 된다.

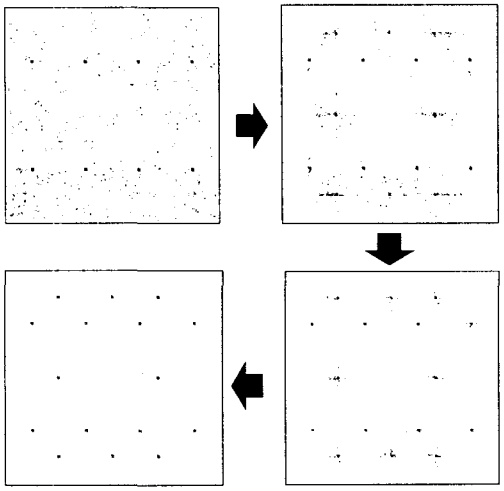
아래 [그림 6]은 실시간 이동객체 데이터 생성기의 실행화면을 나타낸 것이다.



[그림 6] 실시간 이동객체 데이터 생성기

그림의 왼쪽 영역은 한 면이 25.6km인 GALIS의 매크로-셀을 나타낸 것이며, 오른쪽은 이 매크로-셀에 속해있는 특정 마이크로-셀 부분을 보여준다. 매크로-셀의 검은 사각형 영역은 학교 및 회사의 위치를 나타낸 것이며, 위치의 변경이 가능하다.

아래 [그림 7]은 각각 40개의 Student, Faculty, DeskDuty, Outside Duty 이동객체가 등교 및 출근 시나리오에 따라 움직이고 있는 모습을 나타낸 것이다.



[그림 7] 이동객체의 등교 및 출근 움직임 시나리오의 예

4. GALIS의 마스터 LDP 설계 및 구현

4.1 마스터 LDP의 개요

마스터 LDP는 크게 Update_TMO, IndexBuilder_TMO, QueryProc_TMO로 구성되어 있으며, 각 TMO들은 RMMC로 연결되어있다[1][5].

SLDS로부터 전송된 이동객체의 움직임 정보는 우선 Update_TMO내에 위치한 큐에 저장되고, Update_TMO의 Location_DB_Update_SpM 함수에 의해 호출되어, 마스터 LDP내의 영역이라면, IndexBuilder_TMO객체에 의해 로컬 데이터베이스에 저장된다. 그렇지 않다면, RMMC를 통해 LLDS내의 다른 워커 LDP들로 이동객체의 데이터를 보내어 해당 영역의 LDP가 데이터를 저장한다[6].

각 LDP의 IndexBuilder_TMO는 또한 각각의 TimeZone에 따른 위치 데이타 레코드 테이블을 유지한다.

QueryProc_TMO는 SLDS로부터 요청되는 쿼리를 처리하기 위한 TMO이다. SLDS로부터 쿼리를 받은 마스터 LDP의 QueryProc_TMO는 필요에 따라 Query_PickupDecompose함수를 사용하여 서브 쿼리들로 분해한 뒤, 워커 LDP들로 분해된 서브 쿼리들을 전송하고, 그 결과를 Resemble-Results함수로 종합하여 SLDP의 마스터 SDP로 전송한다.

4.2 구현 환경

본 마스터 LDP 프로그램은 펜티엄 III-600 MHz, Ram 512 MByte의 컴퓨터에서 Visual Studio .Net을 사용하여 구현하였다. LDP와 연동되는 상용 데이터베이스 엔진으로는 MicroSoft사의 SQL Server 2000을 사용하였으며, 데이터베이스에 저장되는 테이블 스키마는 아래 [그림 8]과 같다.

```
OID      as  int    // 이동객체의 ID
POS_X    as  int    // 이동객체의 X 좌표
POS_Y    as  int    // 이동객체의 Y 좌표
CELL_NO  as  int    // 이동객체의 해당 셀 번호
UPDATE_TIME  as  datetime // 이동객체가 갱신된 시간
```

[그림 8] 데이터베이스의 테이블 스키마

4.3 Update_TMO의 설계 및 구현

본 논문에서 구현된 Master LDP 프로그램은 Update_TMO 객체와 실시간 이동객체 데이터 생성기가 연결되어 있으며, Location Report Queue를 Linked List형태로 구현하여 동적 할당을 가능하게 한 것이 특징이다.

Update_TMO는 TMO 프로그래밍 스킴을 이용하여 설계되었으며, 크게 TMO 생성자, SpM 함수 부분으로 나뉜다. TMO 생성자는 Update_TMO를 TMO 미들웨어에 등록하는 부분으로서 특히 TMO 등록시에 생성되는 AAC객체는 TMO 미들웨어에 함께 등록되어 주기적으로 UpdateTMO객체의 SpM함수를 호출하는데 이용된다. SpM함수는 TMO 미들웨어에 의해 주기적으로 호출되는 함수로서 본 논문의 Update_TMO의 SpM함수의 동작 알고리즘은 아래 [그림 9]와 같다.

```
[Algorithm 1] Update_TMO.SpM_Update_TMO
  for each MovingObject in Master LDP
     MovingObject.move()
     if MovingObject.ChangeCell is true
        Push MovingObject Data in Location Reports Queue
  call IndexBuilder_TMO.SvM_IndexBuilder_TMO()
[End of Algorithm1]
```

[그림 9] Update_TMO의 SpM_Udpate_TMO 알고리즘

4.4 IndexBuilder_TMO의 구현

IndexBuilder_TMO는 이동객체의 정보를 로컬 데이터 베이스에 저장하는 TMO이다. IndexBuilder_TMO는 크게 TMO 생성자, SpM함수, SvM함수 부분으로 나눌 수 있다. TMO 생성자의 기능은 위 Update_TMO와 같으나, IndexBuilder_TMO의 경우는 TimeZone2~4에 대응하는 저장함수인 SpM_IndexBuilder_TMO_TimeZone_2~SpM_IndexBuilder_TMO_TimeZone_4가 필요하며, 이를 위해 3개의 AAC 객체가 추가된다. IndexBuilder_TMO의 SpM함수들는 주기적으로 호출되어 조건에 맞는 이동객체 데이터를 해당 데이터베이스 테이블에 저장한다. 아래 [그림 10]은 IndexBuilder_TMO의 TimeZone2를 위한 SpM저장 함수의 알고리즘을 기술한 것이다.

IndexBuilder_TMO의 SvM 함수는 Update_TMO의 SpM함수에 의해 호출되어 Location Report Queue에 저장된 이동객체 데이터를 데이터베이스의 해당 테이블에 저장한다.

```
[Algorithm 2.] IndexBuilder_TMO.
                SpM_IndexBuilder_TMO_TimeZone_2
  for All MovingObjects in Table Obj_Loc_TimeZone1
    TempOnject_1 <- First Record of MovingObject
    Store TempObject_1 in Table Obj_Loc_TimeZone2
    TempObject_2 <- Second Record of MovingObject
    while ( Records exist )
      Caculate Distance of Two Objects
      if Distance of Two Objects >= 400
        Store TempObject_2 in Table Obj_Loc_TimeZone2
        TempObject_1 <- TempObject_2
        TempObject_2 <- Next Record of MovingObject
      else
        TempObject_2 <- Next Record of MovingObject
[End of Algorithm 2]
```

[그림 10] IndexBuilder_TMO의 SvM_IndexBuilder_TMO_TimeZone_2

[그림 11]은 본 프로그램에 의해 SQL Server 2000상에 저장되어진 테이블의 예를 나타낸 것이다.

클라이언트로부터 이동객체에 대한 질의가 있을 경우에는 우선 클라이언트의 질의가 SQL 구문으로 변환되며, QueryProc_TMO에 의해 SQL 구문이 처리된 후 결과를 클라이언트에게 통보한다.

	OID	POS_X	POS_Y	CELL_NO	UPDATE_TIME
1	1	18878	3782	36529	2003-08-14 14:19:48.000
2	2	1832	1	520	2003-08-14 14:19:48.000
3	3	6700	1399	8283	2003-08-14 14:19:48.000
4	3	6672	1407	8284	2003-08-14 14:20:00.000
5	3	6699	1384	8281	2003-08-14 14:20:05.000
6	4	25183	17085	61134	2003-08-14 14:19:48.000
7	4	25195	17116	61135	2003-08-14 14:20:05.000
8	5	3597	15826	18782	2003-08-14 14:19:48.000
9	5	3628	15799	18801	2003-08-14 14:20:00.000
10	5	3639	15835	18804	2003-08-14 14:20:05.000
11	6	13460	19597	53293	2003-08-14 14:19:48.000
12	6	13419	19642	53304	2003-08-14 14:20:00.000

[그림 11] SQL Server 2000의 저장 테이블의 예

5. 결론 및 향후 과제

위치 기반 서비스에 있어서 대량의 이동객체를 처리할 수 있는 시공간 데이터베이스의 중요성이 부각됨에 따라 많은 관련 연구가 진행되고 있다. GALIS 아키텍쳐 또한 실시간으로 이동객체를 저장, 관리하기 위한 시공간 데이터베이스 시스템의 하나로 제안된 시스템이다.

본 논문에서는 GALIS 아키텍쳐의 마스터 LDP 부분을 실제로 구현하였으며, 마스터 LDP내의 Update_TMO, IndexBuilder_TMO 및 그 매개함수들을 설계하고, 구현 알고리즘을 제안하였다.

마스터 LDP의 각 클래스는 TMO 프로그래밍 스킴을 사용하여 구현되었으며, TMO 미들웨어와 연동하여 실행되게 된다.

향후 연구과제로는 GALIS의 워커 LDP 부분을 구현하여 TMO 프로그래밍 스킴의 RMMC로 연결하고, 현 시스템 상에서 구현되지 않은 QueryProc_TMO부분을 완성하여야 하며, GALIS와 여타 시공간 데이터베이스 시스템간의 성능비교가 필요할 것이다.

참 고 문 헌

[1] 나연묵, K. H. (Kane) Kim, 왕태영, 김문희, 이종훈, 양영 규, "GALIS : LBS 시스템의 클러스터 기반 신축성 소유 아키텍쳐", 데이터베이스 연구 논문지, Vol 18, No. 4, 2002년 12월, pp 66-80.

[2] Y.Nah, M.H Kim, T.Wang, K.H Kim, Y.K Yang, "TMO-structured Cluster-based Real-time Management of Location Data on Massive Volume of Moving items", STFES 2003, Hakodate, Japan.(to be published)

[3] M.H Kim, T.Wang, K.H Kim, Y.Nah, Y.K Yang, "Distributed Adaptive Architecture for Managing Very Large Volume of Moving Items", IDPT 2003, Beijing, china, (to be published)

[4] 박원순, 전세길, 나연묵, "대용량 이동객체의 위치 정보 인덱싱", 정보과학회 추계 학술발표 논문집, Vol 29, No. 2, 2002년 10월, pp 49-51.

[5] K.H Kim, "Object-Oriented Real-Time Distributed Programming and Support Middleware", ICPADS 2000 (7th Int'l Conf. on Parallel & Distributed Systems), Iwate, Japan,July 2000, pub. by IEEE CS Press (Keynote paper), pp 10-20.

[6] K.H Kim, "APIs Enabling High-Level Real-Time Distributed Object Programming", IEEE Computer, June 2000, pp 72-80