모바일 서비스를 위한 메인 메모리 기반 공간 데이터 관리자

Main-Memory Based Spatial Data Manager for Mobile Service

오병우* / Byoung-Woo Oh

요 약

최근 모바일 환경의 급격한 발전으로 모바일 기기에서 지도를 표현하기 위한 필요성이 급증하고 있다. 본 논문은 모바일 환경에서 공간 데이터를 효율적으로 제공하기 위해 가장 중요한 역할을 담당하는 공간 데이터 관리자를 제안한 논문이다. 본 논문에서 제안하는 공간 데이터 관리자는 상호운용성을 지원하면서도 효율적인 성능을 제공하기 위하여 다양한 방법을 사용한다. 상호운용성을 지원하기 위해서는 국제 표준을 준수하여 상이한 환경에서도 재사용성을 보장한다. 성능 향상을 위해서는 물리적인 메인 메모리에 공간 데이터를 적재하고 서비스하여 디스크 접근 시간을 없애고, 결과 데이터 형식으로 미리 변환한 공간 데이터를 메인 메모리에 적재하여 형식 변환에 걸리는 시간이 필요 없도록 한다. 그리고, 모바일 환경에서 유무선 온라인으로 공간 데이터를 실시간으로 전송하는 경우에 전체 데이터를 전송하지 않고 부분적으로 전송된 데이터만으로도 표현이 가능하도록 하여 응답 시간 및 처리 시간을 최소화할 수 있도록 한다.

Abstract

This paper proposes an efficient spatial data manager for map services in mobile environment. It is designed to provide interoperability and efficient performance at once. To provide interoperability and reusability, the spatial data manager conforms to international standards such as the OpenGIS Simple Features Implementation Specification for OLE/COM, OpenGIS Geography Markup Language (GML) Encoding Specification developed by the Open Geospatial Consortium (OGC). The spatial data manger exploits physical main memory using Address Windowing Extensions supported by Microsoft Windows to manage spatial data for efficient performance by reducing time to read data from disk on user's request. The format of the spatial data in main memory is target data (GML) to reduce conversion time from source data to it. Progressive transmission is also provided to reduce latency time by representing only received partial data for mobile environment without waiting whole transmission.

주요어: 공간 데이터, 공간 데이터 관리자, 메인 메모리, GML, 모바일 서비스, 점진 전송

Keywords: Spatial Data, Spatial Data Manager, Main Memory, GML, Mobile Service, Progressive Transmission

■ 본 연구는 금오공과대학교 학술연구비에 의하여 연구된 논문임.

■ 논문접수 : 2006. 4. 4 ■ 심사완료 : 2006. 4. 19

* 교신저자 금오공과대학교 컴퓨터공학부 조교수(bwoh@kumoh.ac.kr)

1. 서 론

최근 들어 컴퓨터의 초소형화 및 무선 통신의 범용화에 따라 모바일 환경이 급격히 발전하고 있다. 모바일 환경에서 이동중인 사용자는 유무선 네트워크를 통해 원격 서버에 저장되어 있는 공간 데이터를 전송받아 최종적으로 현재의 위치와 관련된 서비스를 제공받을 수 있다 [1, 2, 3, 4, 5]. 공간 데이터를 전송하는 방법은 서비스에 따라 전체 데이터 다운로드 및 질의 검색 결과 전송 두 가지로 나누어 생각할 수 있다.

전체 데이터 다운로드는 주행 안내 서비스 등과 같이 서비스에 사용되는 공간 데이터를 모바일 장치에 저장하고 있는 경우에 전체 공간 데이터를 전송받거나 바뀐 일부분의 데이터를 전송받아서 모바일 장치에 저장된 공간 데이터를 최신 데이터로 갱신하는 방법이다. 질의 검색 결과 전송은 주변 지도 검색 서비스 등과 같이 서비스에 사용되는 공간 데이터를 모바일 장치에 저장하지 않고 원하는 지역의 공간 데이터 요청에 대한 질의를 서버에 전송하고 그 결과를 전송 받아서 모바일 장치에 출력하는 방법이다.

두 가지의 공간 데이터 전송 방법은 모두 신속한 전송 속도를 필요로 한다. 전체 데이터 다운로드는 일반적으로 공간 데이터의 용량이 크므로 전송에 걸리는 시간을 최소화하기 위하여 효율적인 공간 데이터 서버가 필요하다. 그리고, 온라인으로 질의 검색 결과를 전송받는 경우에도 서비스의 성능과 직결되므로 서버에서의 효율적인 공간 질의 처리가 필요하다 [6].

본 논문에서는 모바일 환경에서 이동중인 사용자의 위치 및 위치와 관련된 정보를 표현하기 위해 필수적인 공간 데이터의 효율적인 제공을 위해 가장 중요한 역할을 담당하는 공간 데이터 관리자를 제안한다. 본 논문에서 제안하는 공간 데이터 관리자는 상호운용성 및 효율적인 성능을 모두 지원하기 위하여 각각 다음과 같은 방법을 사용한다.

공간 데이터의 상호운용성과 재사용성을 제공하기 위해서는 OpenGIS Simple Features Implementation Specification for OLE/COM [7], Geographic Information – Geography Markup Language [8] 등과 같은 국제 표준을 준수한다. [7]은 공간 데이터의 다양한 저장 형식에 상관없이 디스크에 저장된 공간 데이터베이스에 접근할 수 있도록 해주는 인터페이스에 대한 표준이다. GML[8]은 공간 데이터를 주고받을 때의 상호운용성 및 재사용성을 제공하기 위한 교환 포맷에 대한 XML 인코딩 표준으로서 WFS[2]의 결과 데이터 포맷으로 사용된다.

모바일 서비스의 성능을 향상시키기 위해서는 물리적인 메인 메모리 사용 [9], 미리 변환된 결과 데이터 (Target Data) 관리, 점진 전송 (Progressive Transmission) 적용과 같은 방법을 사용한다. 공간 데이터 서버의 응답 시간에서 가장 많은 시간이 소요되는 디스크 I/O를 줄이기 위하여 디스크 접근은 물론이고 운영체제에 의한 스와핑도 발생하지 않는 물리적인 메인 메모리를 사용한다. 디스크에 저장된 공간 데이터 포맷을 클라이언트에게 반환할 결과 데이터 포맷으로 변환하기 위한 비용도 성능을 좌우하는 중요한 요소가 되므로 성능향상을 위하여 결과 데이터 포맷으로 미리 변환된 공간 데이터를 관리하는 방법을 사용한다. 온라인으로 질의 검색 결과를 전송받는 경우에 일부의 공간 데이터만을 전송함으로써 전송량을 줄이기 위한 방법으로 점진 전송을 사용한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 메인 메모리 처리 기법에 대해 설명한다. 3장에서는 상호운용성을 제공하는 공간 데이터 관리자의 구성에 대해 살펴보고, 4장에서는 성능향상을 위해 사용한 방법들에 대해 설명한다. 5장에서는 성능에

대한 실험을 통해 본 논문에서 제안하는 메인 메모리 기반 공간 데이터 관리자의 우수성을 증명하고, 마지막으로 6장에서 결론 및 향후 연구방향에 대해 논의한다.

2. 관련 연구

컴퓨터의 발달과 더불어 소프트웨어도 점점 복잡해지고 있다. 복잡하면서도 데이터 집약적인 소프트웨어로는 DBMS가 있다. DBMS는 대용량 데이터를 처리함에 있어서 안정성을 보장하면서도 신속한 응답 속도를 제공하기 위하여 성능향상을 위한 다양한 방법을 사용하고 있다. 특히, 응답 속도의 효율성을 가장 많이 저하시키는 디스크 접근 시간을 줄이기 위해서는 근래에 가격이 하락하고 있는 메인 메모리를 사용하는 방법이 사용되고 있다. 특히, MS-Windows 운영체제에서 동작하는 Oracle, SQL Server, DB2 등과 같은 DBMS에서는 이미 물리적인 메인 메모리 관리를 위하여 AWE(Address Windowing Extensions)를 사용하고 있다 [9].

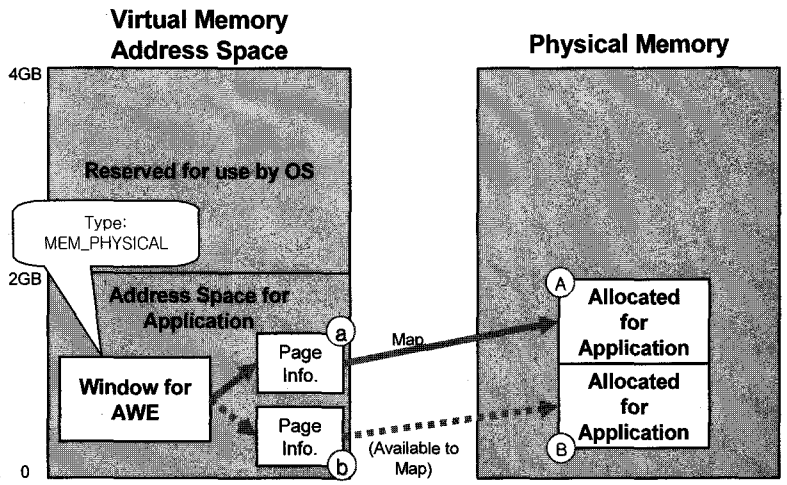
MS-Windows 운영체제에서 대용량의 물리적인 (실제) 메인 메모리 관리를 위한 AWE는 API (Application Programming Interface) 집합으로서 응용 프로그램이 표준 32 비트 가상 주소 체계를 통해 4GB 이상의 메모리 주소(최대 64GB)를 접근할 수 있도록 해준다. 즉, 32 비트 주소 체계에서는 수 표현의 한계인 0 ~ 4,294,967,295 (4G) 사이의 주소를 사용하여야 하지만 AWE는 그 표현 범위내의 가상 주소를 사용하면서도 window라는 방법을 사용하여 4GB 이상의 물리적인 메모리를 사용할 수 있도록 해준다는 것이다. 특히, 운영체제에서는 2GB만을 사용자가 사용할 수 있는 가상 주소로 제공하고 있어서 공간 데이터와 같은 대용량 데이터를 처리하기에는 제한이 될 수 있으므로 메인 메모리를 통한 성능 향상을 위해서는 AWE의 활용이 필수적이다.

AWE에서 사용하는 window는 표준 32 비트 가상 주소 체계를 사용하는 메모리 영역이다. 이 window 메모리 영역은 운영체제에 의해 임의의 메모리(또는 디스크)로 매핑되는 것이 아니라 페이징되지 않는 물리적인 메모리(physical non-paged memory)를 응용 프로그램에서 직접 지정하여 매핑함으로써 사용할 수 있게 된다.

응용 프로그램은 우선 AWE를 위해 사용할 window 가상 메모리 영역을 할당받는다. 일반적인 가상 메모리 할당과는 달리 매개변수에 MEM_PHYSICAL을 지정하여야 한다. 그리고, 응용 프로그램에서 필요한 크기의 물리적인 메모리도 할당 받는다. 컴퓨터에 장착된 물리적인 메모리 크기 이상 또는 최대 64GB 이상은 할당받을 수 없다. 가상 메모리와 물리적인 메모리를 모두 할당받았다면 두 메모리를 매핑하여야 응용 프로그램에서 사용할 수 있다. 즉, window 가상 메모리 영역을 물리적인 메모리로 매핑한 이후에 표준 32 비트 주소 체계를 사용해 읽고 쓰기 연산이 가능하다.

일반적으로는 할당받은 물리적인 메모리의 크기가 가상 메모리 영역보다 크므로 그림 1과 같이 물리적인 메모리의 일부만을 가상 메모리 영역(window)으로 매핑하여 사용하고, 필요에 따라 다른 부분의 물리적인 메모리로 번갈아 가며 매핑하여 사용한다.

그림 1은 MS-Windows 운영체제에서 응용 프로그램을 위한 주소 체계 및 AWE의 작동 방식을 설명한다. 그림에서 보는 바와 같이 window 가상 주소 영역은 매핑되는 물리적인 주소 영역에 대한 정보를 저장하고 있는 페이지 정보 ⓐ를 통해 물리적인 메모리 Ⓐ 부분에 매핑한다. 물리적인 메모리 Ⓑ 부분을 사용하려면 먼저 페이지 정보 ⓑ를 통해 매핑한 후에 window 가상 주소



<그림 1> Address Windowing Extension 개념

에 접근하면 된다. 즉, 동일한 window 가상 주소(예를 들면, 300번지)에 접근하더라도 어디로 매핑되어 있느냐에 따라서 물리적인 메모리 Ⓐ 부분(예를 들면, 시작부터 100번째 상대 주소, 절대 주소 1,100 번지)에 접근할 수도 있고 물리적인 메모리 Ⓑ 부분(예를 들면, 시작부터 100번째 상대 주소, 절대 주소 2,100 번지)에 접근할 수도 있다. 신속한 매핑 처리를 위하여 AWE 사용 방법에는 다음과 같은 제약이 있다.

- 할당된 window 가상 주소 영역은 다른 프로세스와 공유 불가능
- 할당 가능한 물리적인 메모리 크기는 컴퓨터에 설치된 메인 메모리의 물리적인 페이지 수로 제한됨
- 할당된 물리적인 메모리는 lock되어 명시적으로 해제하기 전까지는 다른 프로세스에서 사용 불가능
- 항상 읽기/쓰기 겸용 (읽기-전용 메모리로 활용 불가능)
- 그래픽이나 비디오 버퍼 데이터로 사용 불가능
- 할당된 window 가상 주소 영역을 분리하거나 일부만을 해제하는 것은 불가능

공간 데이터 관리자는 여러 프로세스에서 요청하는 대용량 공간 데이터를 효율적으로 관리하기 위하여 AWE를 사용하여야 하는데 AWE의 제약에 의해 다른 프로세스와의 공유가 불가능하므로 이를 해결하기 위한 방법이 필요하다. 본 논문에서는 공간 데이터를 총괄 관리하는 공간 데이터 관리자를 두고 다른 프로세스에서는 필요시에 공간 데이터 관리자에게 데이터를 요청하게 하는 방법을 사용하였다. 다른 프로세스와는 독립적으로 동작하며 요청을 처리하기 위한 방법으로 COM (Component Object Model) Server(out-of-process)를 사용하였다.

3. 공간 데이터 관리자의 구성

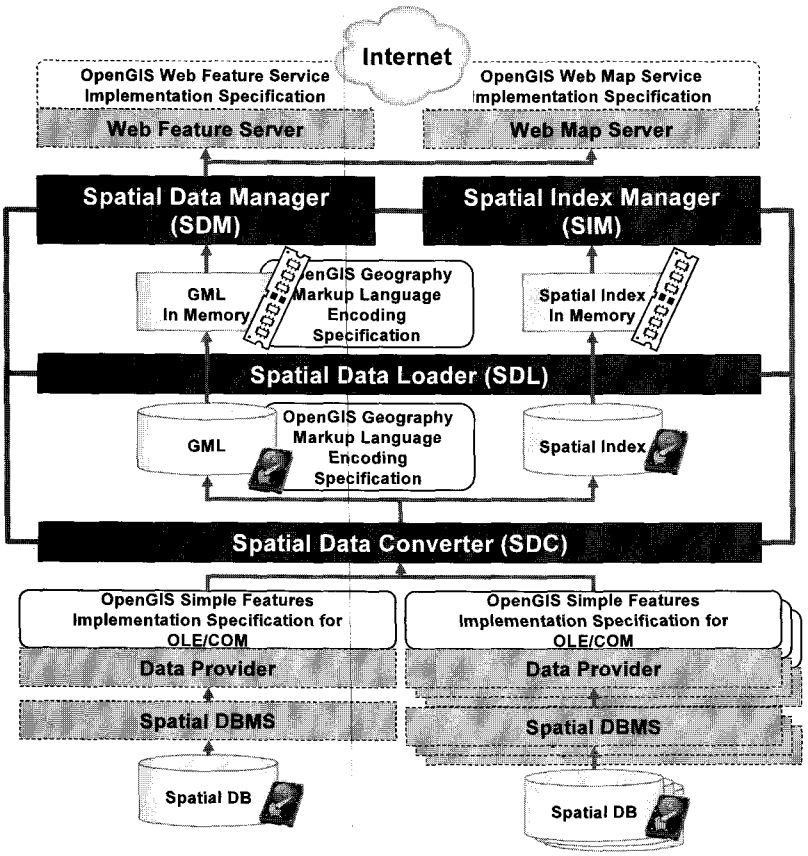
본 논문에서 제안하는 모바일 서비스를 위한 메인 메모리 기반 공간 데이터 관리자는 국제 표준을 준수함으로써 상호운용성 및 재사용성을 지원한다. 본 장에서는 상호

운용성 및 재사용성을 지원하기 위한 공간 데이터 관리자의 구성에 대해 살펴본다.

공간 데이터는 공간 DBMS에 따라서 각자 고유한 형식으로 저장되어 있다. 그러므로, 다양한 공간 데이터를 접근하기 위해서는 각각의 공간 DBMS에서 사용하는 저장 형식에 맞도록 개발하여야 하므로 상호운용성을 저해하는 요인이 되었다. 이러한 단점을 해결하기 위하여 Open Geospatial Consortium (OGC)에서는 공간 데이터 접근에 대한 표준인 OpenGIS Simple Features Implementation Specification을 개발하였다. 본 논문에서는 기존의 데이터베이스 접근 표준 중의 하나인 OLE/DB를 확장한 SFO [7] 표준을 준수하여 공간 DBMS의 종류에 상관없이 다양한 공간 데이터 저장 형식에 접근할 수 있는 상호운용성을 지원한다. 또한, 메인 메모리에서 관리하는 공간 데이터의 형식은 GML[8]을 준수함으로써 공간 데이터의 상호운용성 및 재사용성을 보장한다.

그림 2는 본 논문에서 제안하는 메인 메모리 기반 공간 데이터 관리자의 구성도 및 상호운용성을 위해 사용된 국제 표준을 나타낸다. 그림에서 실선으로 표시된 사각형들이 공간 데이터 관리자의 범위이며 점선으로 표시된 사각형들은 공간 데이터 관리자가 동작하기 위해 필요한 외부 모듈을 나타낸다. 모서리가 둥근 사각형은 상호운용성을 위한 표준을 나타낸다.

그림에서 공간 데이터 관리자의 윗부분에



<그림 2> 메인 메모리 기반 공간 데이터 관리자의 구성도

있는 Web Feature Server (WFS) [2] 및 Web Map Server(WMS)[3]은 인터넷을 통해 사용자의 질의를 받아 해석하는 부분으로서 해석된 질의를 기반으로 응답에 필요한 공간 데이터를 공간 데이터 관리자에게 요청하게 된다. 즉, 공간 데이터 관리자의 입장에서는 검색하는 사용자가 된다.
WFS [2] 표준은 HTTP GET 방식의 질의를 받아서 GML [8] 표준 형식의 공간 데이터를 반환한다. WMS [3] 표준은 HTTP GET 방식의 질의를 받아서 이미지 형태(예를 들면, 확장자가 JPG인 래스터 이미지)의 지도를 반환한다.

본 논문에서 제안하는 공간 데이터 관리자는 특히 WFS에 최적화된 공간 데이터 관리자를 설계하였고 성능향상과 관련된 기술에 대해서는 다음 장에서 자세히 설명한다. WMS에서는 공간 데이터 관리자로부터 질의에 부합하는 공간 데이터를 획득하고 이를 렌더링하여 지도 이미지를 생성할 수 있다.

공간 데이터 관리자의 아래 부분에 있는 Data Provider는 디스크로부터 공간 데이터를 읽어 들이는 역할을 한다. Data Provider는 각각의 공간 DBMS 또는 공간 데이터 저장 형식에 대해서 별도의 구현이 필요하다. 그러나, 일단 개발된 Data Provider가 있다면 사용하는 측에서는 다양한 공간 DBMS에 대해서 동일한 인터페이스로 접근이 가능하다. 이 때 사용되는 표준은 SFO[7]이다.

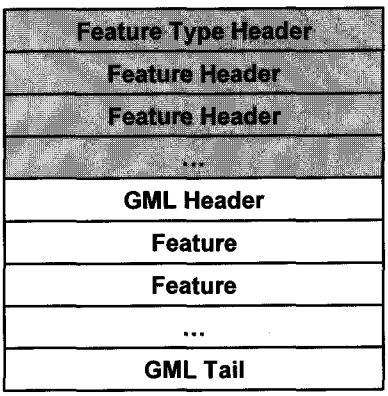
공간 데이터 관리자는 Spatial Data Converter (SDC), Spatial Data Loader (SDL), Spatial Data Manager (SDM), Spatial Index Manager (SIM) 모듈로 구성된다. 다음 절에서는 각 모듈에 대해 자세히 설명한다.

3.1 Spatial Data Converter (SDC)

SDC는 SFO[7] 표준 인터페이스를 통해 디스크로부터 원본 공간 데이터를 읽어서 SDM이 메인 메모리에서 관리하는 공간 데이터 형식으로 변환하는 역할을 담당한다. 효율적인 검색을 위하여 공간 인덱스도 미리 구축하여 파일로 저장한다. 공간 인덱스로는 R*-tree[10]을 사용하며 각 피쳐(feature)에 대한 삽입 연산을 통해 구성된 노드를 디스크에 저장한다.

변환은 피쳐 타입 (feature type) 단위로 처리되고, 변환된 결과는 그림 3과 같은 형식으로 각 피쳐 타입 당 하나의 파일로 저장한다. 그림에서 피쳐 타입 헤더에는 geometry 타입, 전체 공간 데이터의 크기, 피쳐 타입의 전체 MBR (Minimum Bounding Rectangle), 몇 개의 피쳐가 저장되어 있는지 등의 정보가 저장된다. 피쳐 헤드에는 GML 부분에서 해당 피쳐가 위치하고 있는 상대 주소, 피쳐가 몇 개의 점(vertices)으로 구성되어 있는지 등의 정보가 저장된다. GML 부분은 피쳐 타입 전체를 요청할 때 서비스되는 GML 데이터로서 XML 헤더 및 모든 피쳐 데이터를 포함하고 있다.

SDC에서 미리 공간 데이터를 변환시켜 놓는 이유는 두 가지가 있다. 첫째는, 공간 데이터 서비스 기동 시간을 단축하기 위함이다. 일반적으로 공간 데이터 서비스는 준비된 데이터를 서비스하는 경우가 많으므로



<그림 3> 데이터 파일 구조

미리 서비스할 데이터에 대해 실체화 뷰(materialized view) 형태의 변환을 수행하여 서비스 시작에 드는 비용을 최소화할 수 있도록 한다. 둘째는, 물리적인 메인 메모리를 할당하려면 서비스할 공간 데이터의 크기를 미리 알고 있어야 하기 때문이다. 앞서 2장에서 살펴본 바와 같이 물리적인 메모리가 컴퓨터에 충분히 설치되어 있더라도 AWE의 window 가상 주소 영역 할당에는 한계가 있으므로 본 논문에서는 각 피쳐 타입마다 필요한 크기만큼의 물리적인 메모리를 할당하여 관리하고 필요시에 window 가상 주소 영역의 일부를 매핑하여 사용한다.

3.2 Spatial Data Loader (SDL)

SDL은 SDC가 미리 변환해 놓은 공간 데이터 및 공간 인덱스 데이터를 디스크로부터 읽어서 SDM에게 전달하는 역할을 담당한다. 사용자가 변환된 공간 데이터 중에서 서비스할 피쳐 타입을 선택하면 SDL은 선택된 피쳐 타입들에 대해서 파일 시스템으로부터 공간 데이터 및 공간 인덱스 데이터 파일을 읽어서 SDM에서 관리할 수 있도록 해준다. SDM의 물리적인 메모리 할당을 위하여 SDC에 의해 변환된 파일의 크기를 얻는 역할도 수행한다.

SDM은 COM Server (out-of-process) 형태로 구현되어 있어서 독립적인 프로세스로 실행되지 못하므로 SDL에서 SDM을 실행시켜 주어야 한다. COM Server는 운영체제에 의해 관리되므로 SDM을 사용하는 프로세스가 없으면 자동으로 실행을 멈추고 해제된다. 그러므로, SDL은 SDM의 초기화 메소드를 사용함으로써 SDM이 실행되도록 해줄 뿐만 아니라 서비스 개시 이후에 바로 종료하지 않고 사용자가 서비스를 종료할 때까지 계속 실행하고 있어서 SDM이 운영체제에 의해 자동으로 해제되지 않도록 해준다. SDL에 의해 SDM 서비스의 시작과 종료를 제어할 수 있게 되므로 사용자(administrator)의 입장에서 보면 마치 SDL이 서버인 것으로 생각될 수도 있다.

3.3 SDM: Spatial Data Manager

SDM은 공간 데이터를 메인 메모리에 적재하고 사용자의 요구에 따라 공간 데이터를 제공하는 가장 중요한 역할을 담당한다. SDM의 사용자는 크게 적재하는 사용자와 검색하는 사용자의 두 종류로 나뉜다. 적재하는 사용자로는 메인 메모리에 공간 데이터를 적재하는 SDL이 있다. 검색하는 사용자로는 이미 메인 메모리에 적재된 공간 데이터에 대한 접근을 요구하는 클라이언트로서 WFS가 있다. 검색하는 사용자는 먼저 SDM이 공간 데이터를 메인 메모리에 적재하고 있는지 여부를 검사한 다음에 적재하고 있을 경우에만 검색을 요청하여야 한다.

SDM은 앞서 2장에서 설명한 AWE를 사용하여 대용량의 물리적인 메인 메모리를 관리한다. 먼저 미리 변환된 공간 데이터 파일 및 공간 인덱스 파일의 크기를 얻어서 피쳐 타입별로 필요한 메인 메모리의 크기를 계산한다. "공간 데이터 파일 크기 + 공간 인덱스 파일 크기" 의 계산식으로 계산이 가능하다.

피쳐 타입을 위한 크기가 산출되면 그 크기만큼의 물리적인 메인 메모리를 할당한 후에 공간 데이터 파일 및 공간 인덱스 파일을 메인 메모리로 적재한다. 예를 들어, 100개의 피쳐 타입을 서비스하기 위해서는 각자 다른 크기(피쳐 타입별로 필요한 크기)로 100번의 물리적인 메인 메모리 할당이 수행된다. 물리적인 메인 메모리 접근에 필요한 window 가상 주소 영역은 가장 크기가 큰 피쳐 타입의 크기만큼 할당하면 된다. 그리고, 특정 피쳐 타입에 접근하려면 해당 피쳐 타입이 저장된 물리적인 메인 메모리의 페이지 정보를 통해

window 가상 주소 영역을 매핑한 후 사용한다. 즉, n개의 피쳐 타입에 접근하려면 n번의 매핑이 필요하다.

피쳐 타입별로 저장된 공간 데이터 파일 및 공간 인덱스 파일은 모두 상대 주소가 0번지부터 시작하도록 변환되어 있으므로 디스크로부터 읽어서 물리적인 메인 메모리에 적재할 때 추가 변환 없이 그대로 메인 메모리에 적재하고 서비스할 수 있다.

3.4 SIM: Spatial Index Manager

SIM은 질의에 따라 공간 데이터를 효율적으로 검색하는 역할을 담당한다. 본 논문에서는 R*-tree를 메인 메모리에서 구현한 구조를 사용하여 공간 데이터를 검색한다. SIM은 크게 SDC에서 사용하는 부분과 SDM에서 사용하는 부분으로 구분된다.

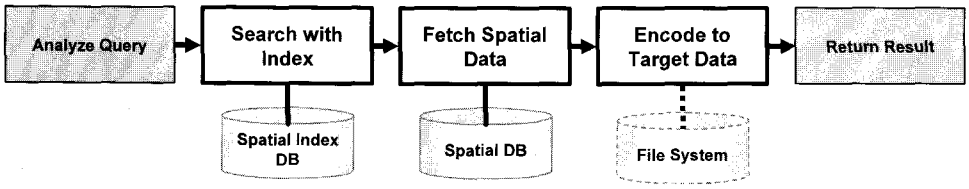
SDC에 의해 사용되는 부분은 피쳐를 삽입하면서 R*-tree의 노드 구조를 구성하는 부분이다. SDC가 임의의 피쳐 타입에 속한 피쳐들을 변환하면서 피쳐의 공간 데이터를 R*-tree에 삽입하면서 노드 구조를 구성한다. 노드 구조는 디스크에 파일로 저장된다. 공간 인덱스 파일에서 다른 노드를 참조하는 포인터는 노드의 순번에 따른 노드 번호를 사용하므로 "노드 번호 * 노드 크기"로 파일상의 위치를 계산할 수 있다. 공간 인덱스 파일을 추가 변환 없이 물리적인 메인 메모리로 적재한 후에도 "공간 인덱스 시작 주소 + 노드 번호 * 노드 크기"의 계산식으로 노드의 위치를 쉽게 계산할 수 있다.

SDM에서 사용하는 부분은 SDL이 디스크로부터 읽은 노드 구조를 물리적인 메인 메모리에 적재하는 기능과 공간 데이터를 검색하는 부분이다. SDM에게 공간 데이터 검색을 요청하는 사용자(예를 들면, WFS 등)는 영역 질의를 사용할 수 있다. SDM은 영역 질의에 대한 결과를 검색하기 위해 SIM을 사용한다. SIM에서는 메인 메모리에 적재된 R*-tree 노드 구조를 기반으로 검색을 수행하고 질의에 부합하는 피쳐들의 리스트를 반환한다. 리프 노드에서 피쳐를 참조하는 포인터는 공간 데이터 파일(GML)에서의 피쳐의 순번에 따른 피쳐 번호를 사용한다. 그러므로, 메인 메모리에 적재된 피쳐 데이터에 접근하기 위해서는 "피쳐가 위치하고 있는 상대 주소 [피쳐 번호]"의 계산식으로 피쳐의 위치를 계산할 수 있다. SDM에서 질의에 대한 결과를 작성하는 과정은 다음과 같다. 먼저 GML의 헤더 부분을 복사하고, SIM에 의해 반환된 피쳐 리스트에 속한 피쳐들의 GML 데이터를 차례대로 복사한 후, 최종적으로 GML의 종료 부분을 복사한다.

4. 성능 향상 기법

모바일 환경은 공간상에서 이동중인 사용자의 위치와 밀접한 관련이 있으므로 공간 데이터의 사용이 필수적이다. 모바일 환경에서 활용할 수 있도록 공간 데이터를 서비스하는데 있어서 성능에 가장 중요한 부분은 공간 데이터의 효율적인 관리를 담당하는 부분이다. 본 논문에서는 기존의 전형적인 공간 데이터 관리자의 단점을 극복하고 효율적인 성능을 보장하기 위하여 몇 가지 기법을 사용하였다.

전형적인 공간 데이터 서버는 그림 4와 같은 과정을 거쳐 서비스를 제공한다. 공간 데이터 서버는 우선 사용자의 질의를 분석하고, 디스크에 저장된 공간 인덱스 DB를 사용하여 질의에 부합하는 공간 데이터를 검색한다. 질의를 만족하는 검색 결과가 있다면 해당하는 공간 데이터를 디스크로부터 메인 메모리로 읽어 들인다. 디스크로부터 읽은 데이터는 결과 데이터로는 부적합하므로 결과 데이터 형식에 맞도록 인코딩 과정이 필요하다. 이 과정에서 간혹 파일 시스템



<그림 4> 전형적인 공간 데이터 서버의 처리 절차

을 사용하여 디스크에 결과 데이터를 임시 저장하는 경우도 있다. 인코딩 시에는 사용자의 요구에 맞는 데이터 형식으로 변환하여야 하는데 상호운용성 및 재사용성을 위해서는 GML과 같은 국제 표준을 준수하는 것이 바람직하다. 마지막으로, 완성된 결과를 질의 요청 사용자에게 반환한다.

전형적인 공간 데이터 서버의 단점은 공간 인덱스 검색 및 공간 데이터 읽기 과정에서 발생하는 디스크 접근과 결과 데이터 형식에 맞도록 변환하는 인코딩 과정에서 발생하는 시간 지연이다. 본 논문에서는 이러한 전형적인 공간 데이터 서버의 단점을 극복하고 성능을 향상시키기 위해서 디스크 접근과 인코딩에 소모되는 시간을 없앨 수 있는 다음과 같은 기법을 사용한다.

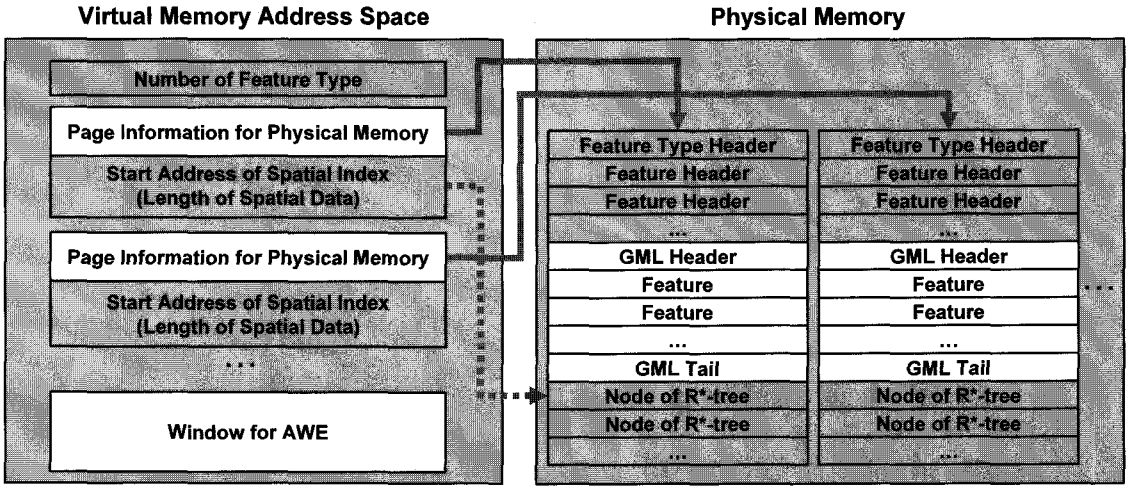
공간 인덱스 검색 및 공간 데이터 읽기 과정에서 상대적으로 응답 시간이 늦은 디스크 대신에 메인 메모리를 사용하고, 인코딩 과정에서 발생하는 시간을 최소화하기 위하여 미리 결과 데이터 형식으로 변환하는 방법을 사용한다. 그리고, 결과 전송 과정에서 발생하는 시간을 줄이기 위해서는 한번에 전송 결과를 모두 보내는 것이 아니라 일부만을 우선 전송하고 나머지 부분을 사용자의 요구에 따라 점진적으로 전송하는 방법도 제공한다.

4.1 물리적인 메인 메모리 사용

본 논문에서는 서비스할 공간 데이터를 효율적으로 관리하기 위하여 메인 메모리를 활용한다. 메인 메모리를 사용함으로써 얻을 수 있는 장점은 디스크의 I/O 시간을 배제하여 신속한 공간 데이터 서비스가 가능하도록 하는 것이다. 그러나, 메인 메모리는 운영체제에 의해 관리 되므로 메모리 부족 시에도 스와핑이 발생하지 않는 물리적인 메인 메모리를 사용하려면 해당 운영체제에서 지원하는 방식을 따라야 한다. 본 논문에서는 MS-Windows 운영체제에서 지원하는 AWE 방법을 사용하여 스와핑 또는 페이징이 발생하지 않는 실제 물리적인 메인 메모리에서 공간 데이터를 관리하고 서비스하여 성능 향상을 도모한다.

본 논문에서 제안하는 공간 데이터 관리자는 피쳐 타입별로 미리 변환된 공간 데이터 파일 및 공간 인덱스 파일을 물리적인 메인 메모리에 적재하고 관리한다. 그림 5는 공간 데이터 관리자가 가상 메모리와 물리적인 메인 메모리를 관리하는 구조를 나타낸다.

가상 메모리는 응용 프로그램에서 일반적으로 사용할 수 있는 메모리로서 공간 데이터 관리자는 각 피쳐 타입별 정보 및 AWE를 위한 window 가상 주소 영역을 할당하기 위해 사용한다. 물리적인 메인 메모리는 서비스할 피쳐 타입들에 대한 공간 데이터 및 공간 인덱스 구조를 저장하기 위해 사용한다. 각각의 피쳐 타입은 별도의 물리적인 메인 메모리 공간에 할당하고 window 가상 주소 영역으로 매핑하기 위한 페이지 정보



<그림 5> 메인 메모리 관리 구조

는 가상 메모리에 할당한다.

물리적인 메인 메모리에 할당되는 피쳐 타입별 크기는 미리 변환된 공간 데이터 파일 크기와 공간 인덱스 파일 크기의 합으로서 피쳐 타입마다 상이하다. 그러므로, 피쳐 타입의 페이지 정보 크기도 상이하게 된다. 본 논문에서는 AWE를 위한 window 가상 주소 영역의 크기는 서비스할 피쳐 타입의 할당 크기 중 가장 큰 크기만큼을 할당한다. 특정 피쳐 타입에 접근하려면 해당 피쳐 타입의 페이지 정보를 통해 window 가상 주소 영역을 해당 피쳐 타입이 실제 저장된 물리적인 메인 메모리에 매핑해 놓고 window 가상 주소 영역을 접근하면 된다.

4.2 미리 변환된 결과 데이터 관리

본 논문에서는 클라이언트가 요청하는 결과 데이터 형식에 맞춰서 공간 데이터를 미리 변환해 놓고 결과 데이터 형식을 공간 데이터 관리자에서 관리함으로써 원본 데이터 형식을 결과 데이터 형식으로 요청시마다 변환하는 시간 지연 없이 신속한 서비스를 제공하도록 해준다. 결과 데이터 형식으로는 상호운용성과 재상용성을 제공할 수 있도록 국제 표준인 GML을 채택하였다. 일반적으로 인터넷을 통해 GML을 제공하려면 WFS를 주로 사용한다. WFS는 공간 데이터를 서비스하기 위하여 공간 데이터 관리자를 사용하여야 하는데 본 논문에서 제안하는 메인 메모리를 사용한 공간 데이터 관리자는 WFS의 요구를 모두 수용할 수 있도록 구현하였다. WFS에서 공간 데이터를 얻기 위한 "GetFeature" 연산뿐만 아니라 "GetCapabilities" 및 "DescribeFeatureType" 연산에 대한 결과 데이터도 관리한다. 그러나, 공간 데이터 관리자가 WFS 전용으로 국한되는 것은 아니고 공간 데이터를 GML 형식으로 요청하는 모든 서비스에 범용으로 사용할 수 있다.

공간 데이터는 일반 데이터와는 달리 가변 길이 데이터이다. 결과 데이터의 형식인 GML은 텍스트 데이터인 XML의 형식이므로 특히 가변 길이 데이터 처리에 주의하여야 한다. XML은 Document Object Model (DOM)을 지원하여 엘리먼트를 노드처럼 쉽게 처리할 수 있는 방법을 제공하지만 성능이 낮다. 본 논문에서는 성능을 향상시키면서도 텍스트 데이터에서 가변 길이 데이터

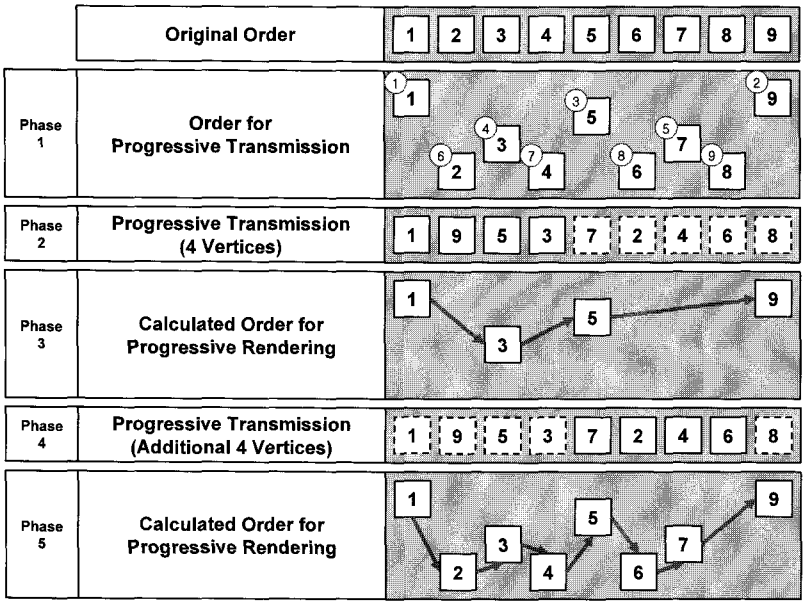
를 처리하기 위하여 각 피쳐에 대한 참조 정보(시작 주소)를 관리한다. 그림 5에서 피쳐에 대한 정보를 저장하는 "Feature Header"는 피쳐의 시작 주소를 포함하고 있어서 순차적으로 할당된 피쳐 번호만 알면 해당 피쳐의 엘리먼트를 접근할 수 있도록 해준다. 피쳐 번호는 공간 인덱스에서 피쳐의 ID로 사용된다. 그러므로, 공간 인덱스를 통해 공간 질의에 대한 결과 리스트를 얻었다면, 리스트에 속한 각각의 피쳐 번호를 기반으로 피쳐 엘리먼트가 위치한 주소를 알 수 있고, 해당 엘리먼트들을 연결하여 공간 질의에 대한 결과 데이터를 생성할 수 있다.

4.3 점진 전송

공간 데이터 관리자를 사용하는 경우 중에서 WFS를 통해 유무선 통신을 통해 온라인으로 공간 데이터를 전송받아 모바일 기기 화면에 출력하는 응용도 가능하다. 일반적으로 모바일 기기는 데스크탑 PC에 비해 상대적으로 성능이 낮고 화면도 작으므로 대용량의 복잡한 공간 데이터 표현에 부적합하다. 특히, zoom-out되어 넓은 범위를 표현하는 지도를 작은 화면에 출력하면 처리 응답 속도도 늦어질 뿐만 아니라 너무 복잡하여 인식하기 어렵게 된다.

본 논문에서는 모바일 서비스를 위한 공간 데이터 전송의 경우에 polyline, polygon 등과 같이 여러 개의 점으로 구성된 geometry의 좌표를 한번에 전부 보내는 것이 아니라 일부만을 전송하고 요청에 따라 나머지 부분을 차례대로 전송할 수 있도록 해주는 점진 전송 방법을 제공한다. 그림 6은 본 논문에서 제안하는 점진 전송의 예제를 보여준다. 그림의 단계 1에서 보는 바와 같이 점진 전송을 위해서는 좌표의 순서를 변환하는 과정이 필요하다. 그 뒤에 클라이언트가 4개의 좌표를 요구하면 단계 2와 같이 변환된 순서에 따라 좌표를 전송한다. 실선으로 표시된 사각형의 좌표가 전송된 좌표이고 점선으로 표시된 사각형의 좌표는 전송하지 않은 좌표를 나타낸다.

클라이언트에서는 단계 3과 같이 좌표의



<그림 6> 점진 전송 예제

순서를 다시 변환한 뒤 화면에 출력하게 된다. 좌표는 순서대로 앞의 두 좌표 사이에 삽입하면 된다. 예를 들어, [1] [9]가 있고 [5]가 온다면 [1]과 [9] 사이에 [5]를 삽입하여 [1] [5] [9]의 순서가 되고, [1] [5]가 있고 [3]이 온다면 [1]과 [5] 사이에 [3]을 삽입하여 [1] [3] [5]를 만들어서, 최종적으로는 [1] [3] [5] [9]를 만들 수 있다.

단계 4는 클라이언트가 추가로 4개의 좌표를 요구한 경우에 이미 전송된 4개의 좌표를 제외하고 실선으로 표시된 사각형의 좌표만을 전송하는 예제를 보여준다. 이를 전송받은 클라이언트는 단계 5와 같이 좌표의 순서를 변환하여 화면에 출력한다. 클라이언트는 이러한 단계를 반복하면서 전체 점의 좌표를 모두 화면에 출력할 수도 있지만, 일부의 좌표만으로도 지도를 인식할 수 있는 상황이라면 굳이 전체 좌표를 화면에 출력하지 않고도 원하는 작업을 수행할 수 있다. 그러므로, 점진 전송 기법을 통해 데이터의 전송량을 줄일 수 있는 장점이 있다.

본 논문에서 제안하는 점진 전송을 위해서는 좌표의 순서를 바꾸어야 하는데 공간 데이터 관리자가 메인 메모리에서 관리하는 공간 데이터가 텍스트 형식의 결과 데이터(GML)이므로 피쳐의 geometry 좌표 데이터를 처리할 때는 주의가 필요하다. 텍스트 데이터에서는 좌표 데이터의 길이가 상이할 수 있다. 예를 들면, "3211.3" 은 6바이트를 사용하고 "33111.30" 은 8바이트를 사용한다. 그러므로, 텍스트 데이터에서 좌표 처리를 용이하게 하려면 길이를 일정하게 하는 것이 바람직하다. 메인 메모리내에서 좌표의 길이를 일정하게 하기 위하여 자리수를 통일된 형식으로 관리함으로써 텍스트를 파싱하지 않고도 다음 점의 좌표를 쉽게 계산 가능하도록 한다. 현재는 국내에서 많이 사용되는 TM 좌표 체계를 처리할 수 있도록 "%10.3f" 형식을 사용한다. 즉, 총 10바이트를 사용하고 소수점 3째 자리까지 표현한다. 예를 들면, "318859.235,544558.638 3 1 9 0 7 7 . 8 3 1 , 5 4 4 4 9 0 . 1 3 5 319299.833,544377.083" 과 같이 x축 좌표와 y축 좌표는 콤마(,)로 구분하고 다음 점의 좌표는 공백으로 구분하여 한 점의 좌표는 22(10 + 1 + 10 + 1)바이트로 일정하게 통일한다. 점의 좌표 길이가 일정하므로 점진 전송을 위해서 점의 순서를 변환할 때 신속한 변환이 가능하다.

5. 성능 평가

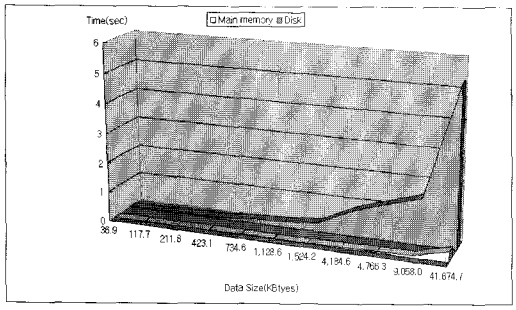
본 논문에서 제안하는 공간 데이터의 관리자의 성능을 평가하기 위하여 디스크로부터 공간 데이터를 읽어서 GML을 생성하는 방법과 비교 실험한다. 그리고, 점진 전송의 렌더링 결과 및 전송량의 변화를 측정한다.

5.1 응답 속도 평가

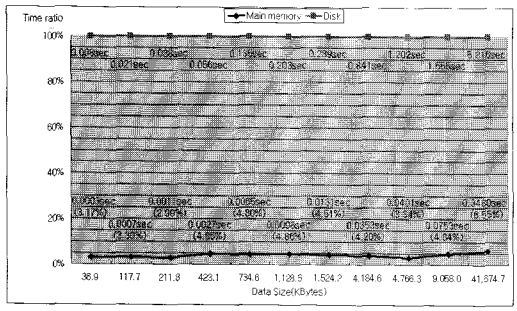
본 절에서는 물리적인 메인 메모리를 사용하는 공간 데이터 관리자의 응답 속도와 디스크에 저장된 공간 데이터를 읽어서 응답할 때의 속도를 비교한다. 실험에서는 공간 데이터 관리자를 통해 메인 메모리에 접근하여 요청을 처리하는 방법과 디스크에 직접 접근하여 공간 데이터를 읽어오고 GML로 변환하는 방법을 사용하였으며 두 방법의 수행시간을 비교하여 본 논문에서 제안하는 방법이 얼마나 효율적인지를 검증하였다.

실험 환경으로 CPU는 AMD Athlon 64 Processor 3000+, 메모리는 삼성 DDR SDRAM PC3200 3GB, 하드디스크는 삼성 S-ATA II 7200rpm 250GBytes 사양의 컴퓨터를 사용하였으며 Windows XP SP2 운영체제에서 실험을 수행하였다. 실험에 사용된 Shape 데이터는 S01~S11의 11개 샘플이며 각 샘플 당 100번을 요청하여 수행시

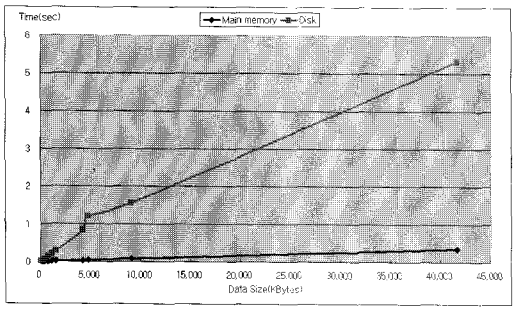
간을 측정하였다. 그림 7은 메인 메모리를 사용하여 요청을 처리하는 방법과 디스크에 접근하여 요청을 처리하는 방법의 성능 비교 결과를 보여준다. 그림의 그래프를 위해 사용된 측정 데이터 원본은 모두 동일하며 다각적인 분석을 위하여 세 종류의 그래프로 표시하였다.



(a) 수행시간



(b) 디스크 대비 메인 메모리 수행시간 비율



(c) 수행시간 증가율

<그림 7> 응답 시간 비교

그림 7(a)에서 보는 바와 같이, 디스크에 접근하는 방법은 공간 데이터의 크기가 커짐에 따라 수행시간이 급격하게 증가하고 있다. 디스크 접근의 경우 41,674.7 KBytes 크기의 공간데이터가 요청되어 처리하는데 5.32초가 소요되었다. 그러나 메인 메모리의 경우는 0.35초가 소요되어 디스크에 접근하는 방법보다 성능이 월등하게 우수함을 알 수 있다. 메인 메모리를 사용하는 방법이 디스크를 사용하는 방법보다 평균적으로 24배 효율적이다.

그림 7(b)는 실험결과를 디스크에 접근하는 방법 대비 메인 메모리 수행시간의 비율을 나타낸 그림이다. 디스크에 표시된 숫자는 수행시간이며 메인 메모리에 표시된 숫자는 수행시간과 디스크 대비 메인 메모리 비율이다. 그림 7(b)에서 보는 바와 같이 디스크에 접근하는 수행시간을 100%로 가정했을 때 메인 메모리를 사용한 방법에서는 평균적으로 디스크에 비해 4.3%의 수행시간만이 소요됨을 확인할 수 있다.

그림 7(c)에서 보는 바와 같이 공간 데이터의 크기가 증가함에 따라 수행시간이 증가하는 비율은 메인 메모리를 사용한 방법이 디스크에 접근하는 방법보다 기울기가 작은 것을 알 수 있다. 그러므로, 공간 데이터의 크기가 크면 클수록 본 논문에서 제안하는 공간 데이터 관리자를 사용하는 방법이 더욱 효율적임을 알 수 있다.

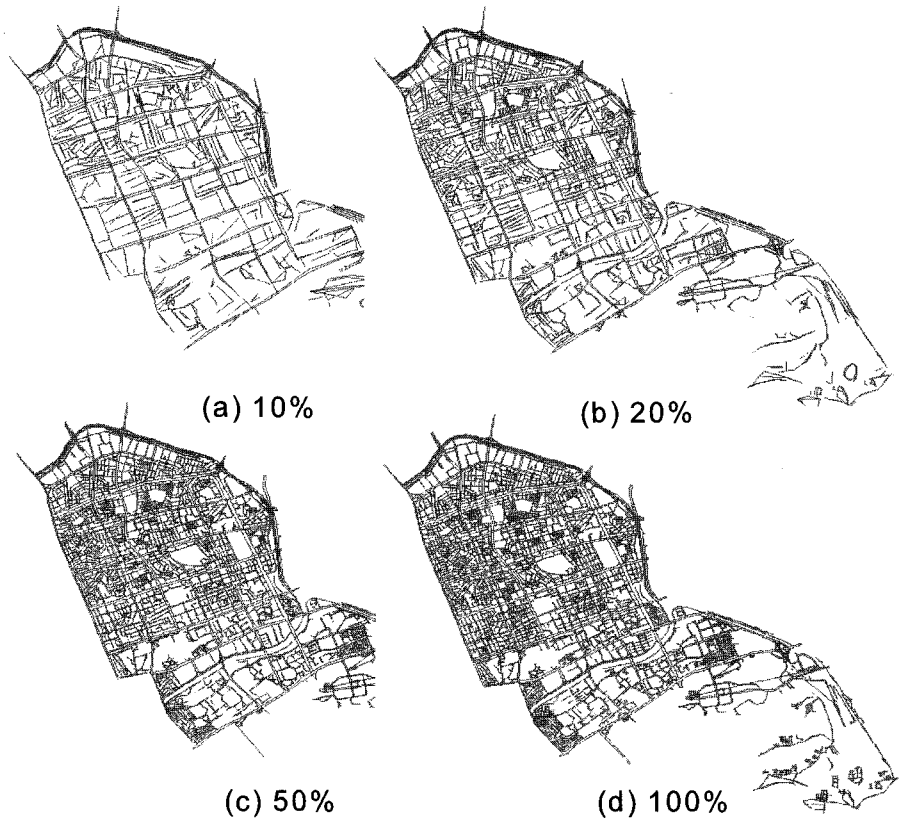
5.2 점진 전송 성능 평가

본 절에서는 점진 전송의 성능을 평가하기 위하여 점진 전송을 통해 공간 데이터를 전송하고 렌더링 했을 때의 결과와 전송되는 데이터의 크기를 측정하였다. 그림 8은 공간 데이터를 요청함에 있어서 점진 전송을 사용하여 각 피쳐별로 geometry를 구성하는 점의 (a) 1/10, (b) 1/5, (c) 1/2, (d) 전체를 전송받아서 렌더링한 결과이다. 모바일

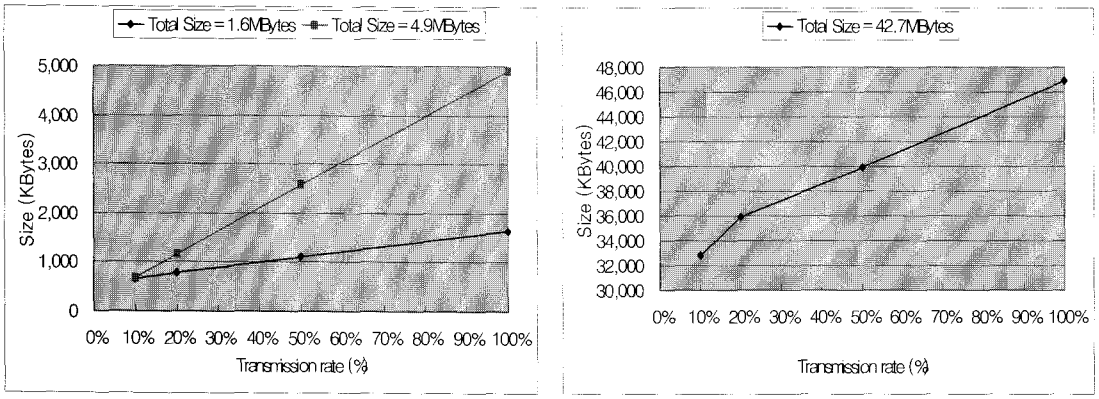
기기의 화면이 일반적으로 작은 점을 고려한다면 굳이 전체 데이터를 모두 전송받지 않고 일부만을 전송받아 표현하여도 전체적인 지형은 파악할 수 있다. 만약, 자세한 지형을 파악하여야 한다면 원하는 지역을 zoom-in하여 전체 데이터를 전송받을 수 있다. 클라이언트에서 zoom의 비율을 가지고 자동으로 점진 전송의 양을 결정할 수도 있고, 자동으로 10%만을 전송받아 표시하였으나 사용자가 원할 경우는 추가의 30%를 전송받아서 전체 데이터의 40%에 해당하는 지도를 표시할 수도 있다.

그림 9는 10%, 20% 50%, 100%의 점진 전송에 사용된 데이터 전송량을 측정하고 이를 기반으로 인터폴레이션한 그래프를 보여준다. 결과 데이터인 GML의 특성상 헤더 부분과 엘리먼트 표현을 위해 기본적으로 사용되는 데이터 크기가 있으므로 y축의 절편이 0이 되지는 않지만 점진 전송에 따른 데이터 전송량이 거의 일정한 기울기를 갖고 증가함을 알 수 있다.

점진 전송에서는 피쳐 타입을 구성하는 피쳐의 개수보다는 피쳐를 구성하는 geometry 점의 개수가 데이터 전송량의 중요한 변수로 작용한다. 즉, 피쳐의 개수가 많고 피쳐를 구성하는 geometry 점의 개수가 적은 피쳐 타입보다는 피쳐의 개수가 적고 geometry 점의 개수가 많은 복잡한 피쳐 타입일수록 점진 전송에 의한 성능 향상을 보다 많이 기대할 수 있다. 그림에서 전체 크기가 4.9메가바이트인 피쳐 타입은 피쳐의 개수가 998개이고 42.7메가바이트인 피쳐



<그림 8> 점진 전송 결과 (Total Size=1.6MBytes)



<그림 9> 점진 전송의 데이터 전송량

타입은 140,958개이다. 크기로는 4.7배의 차이가 나지만 개수의 차이는 141.2배가 되므로 상대적으로 피쳐의 개수가 많다. 50%의 전송량을 살펴보면 전체 크기가 4.9메가 바이트인 피쳐 타입은 68.3%이고 42.7메가 바이트인 피쳐 타입은 85.0%로서 피쳐의 개수 비율이 적은 피쳐 타입이 점진 전송의 효율성이 높음을 알 수 있다.

6. 결 론

본 논문에서는 대용량 공간 데이터를 효율적으로 서비스할 수 있는 메인 메모리 기반 공간 데이터 관리자를 제안하였다. 메인 메모리 기반 공간 데이터 관리자는 다음과 같은 장점을 갖는다. 첫째, 국제 표준을 준수하여 상호운용성 및 재사용성을 제공한다. 둘째, 물리적인 메인 메모리를 통해 공간 데이터를 관리하므로 디스크 I/O 시간을 배제하여 신속한 서비스를 가능하게 한다. 셋째, 서비스할 원본 공간 데이터 형식을 요청시마다 결과 데이터 형식으로 변환하지 않고 미리 변환하여 결과 데이터를 관리함으로써 성능을 향상시킨다. 넷째, 모바일 환경에서 유무선 온라인으로 공간 데이터를 실시간으로 전송하는 경우에 전체 데이터를 전송하지 않고 부분적으로 전송하는 점진 전송 방법을 제공하여 응답 속도를 줄인다. 다섯째, 온라인으로 공간 데이터를 서비스할 경우에 주로 사용되는 WFS를 위해 "GetFeature" 연산뿐만 아니라 "GetCapabilities" 및 "DescribeFeatureType" 연산에 대한 결과 데이터도 관리하여 신속한 서비스를 제공한다. 여섯째, 공간 데이터 관리자를 COM Server로 구현하여 다른 프로세스에서도 자유로이 공간 데이터를 요청할 수 있는 범용성을 제공한다.

제안한 공간 데이터 관리자의 성능을 평가하기 위해서는 실험을 수행하였다. 실험은 메인 메모리를 사용한 공간 데이터 관리자와 디스크로부터 공간 데이터를 읽어 GML을 생성하는 방법의 시간을 비교하였다. 본 논문에서 제안한 공간 데이터 관리자는 디스크를 사용한 방법보다 평균 4.3%의 시간만으로 신속하게 응답을 처리할 수 있었다. 또한, 점진 전송에 따른 렌더링의 결과와 전송량의 변화도 실험하였다. 점진 전송을 통해 전체 데이터의 1/2만을 전송하는 경우 전송량은 최대 52%까지 줄일 수 있었고 피쳐의 개수가 많아서 점진 전송에 적합하지

않은 경우도 85%까지 줄일 수 있었다.

향후 연구방향으로는 모바일 서비스를 위해 공간 데이터를 재구성하는 방법과 재구성된 데이터에 효율적인 공간 인덱스를 개발하여 공간 데이터 관리자의 효율성을 더욱 증대시키는 것이 있다. 또한, 모바일 서비스에 맞는 결과 데이터 형식을 연구하여 그 결과 형식을 공간 데이터 관리자가 메인 메모리에서 관리하도록 하는 것도 가능하다.

참고문헌

1. Buttenfield, B.P., "Transmitting Vector Geospatial Data across the Internet," Proc. GIScience (LNCS 2478), 2002, pp. 51-64.
2. Vretanos, P.A., *Web Feature Service Implementation Specification (version 1.1.0)*, OpenGIS Consortium Inc., 2005.
3. Beaujardiere, J., *Web Map Service Implementation Specification (version 1.3.0)*, OpenGIS Consortium Inc., 2006.
4. Oh, B.W., Kim, M.S., Kim, M.J., and Lee, E.K., "Spatial Data Server for Mobile Environment," Proc. EDBT (LNCS 2992), 2004, pp. 872-874.
5. 김민수, 김미정, 이은규, 주인학, 오병우, "무선 네트워크 환경을 고려한 공간정보 웹 서비스 프레임워크," 개방형지리정보시스템학회 논문지, 제6권, 제2호, 2004, pp. 63-75.
6. Bertolotto, M. and Egenhofer, M., "Progressive Transmission of Vector Map Data over the World Wide Web," Proc. GeoInformatica, Vol. 5, 2001, pp. 345-373.
7. Runnion, E., Beddoe, D., Bacharach, S., Arctur, D., Reilly, J., and Batty, P., *Simple Features Specification For OLE/COM Specification (Revision 1.1)*, OpenGIS Consortium Inc., 1999.
8. Cox, S., Daisey, P., Lake, R., Portele, C., and Whiteside, A., *Geographic Information -Geography Markup Language (version 3.1.0)*, OpenGIS Consortium Inc., 2004.
9. MSDN, *Address Windowing Extensions*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/memory/base/address_windowing_extensions.asp, Microsoft.
10. Beckmann, N., Kriegel, H.P., Schneider, R., and Seeger, B., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. ACM SIGMOD, 1990, pp. 322-331.

오병우

1993년 건국대학교 전자계산학과 졸업 (학사)
1995년 건국대학교 전자계산학과 졸업 (석사)
1999년 건국대학교 전자계산학과 졸업 (박사)
1999년~2004년 한국전자통신연구원
텔레매틱스연구단 선임연구원
2004년~현재 금오공과대학교 컴퓨터공학부 조교수
관심분야: 데이터베이스, GIS, Mobile GIS,
텔레매틱스 등