대용량 궤적 데이터를 위한 효과적인 인덱싱 기법

차창일*, 원정임**, 김상욱**
*(주)에이록스 사업1그룹 개발3팀
**한양대학교 전자컴퓨터통신공학과
e-mail: charose@alox.com, {jiwon, wook}@hanyang.ac.kr

An Effective Indexing Method for Trajectory Databases

Chang-Il Cha*, Jung-Im Won**, Sang-Wook Kim**
*ALOX CO.LTD Buniness Divison1 Development 3 Team
**College of Information and Communications, Hanyang University

요 약

본 연구에서는 대용량 궤적 데이터베이스에서 영역 질의를 효과적으로 처리하기 위한 인덱싱 기법에 대하여 논의한다. 먼저, 기존 인덱싱 기법의 문제점을 지적하고, 이러한 문제점을 해결하는 새로운 기법을 제안한다. 제안된 기법에서는 우선 시간 차원을 다수의 시간 구간으로 분할하고, 인덱싱의 대상이 되는 전체 라인 세그먼트들을 시간 구간별로 구분한다. 각 시간 구간에 속하는 라인 세그먼트들에 대하여 별도의 인덱스를 구축한다. 또한, 디스크에서 관리되는 과거 시간 구간에 대한 인덱스들과는 달리 최근 시간 구간에 대한 인덱스는 메인 메모리상에 관리함으로써 삽입과 검색의 성능을 크게 개선할 수 있다. 각 시간 구간에 속하는 라인 세그먼트들은 다음과 같은 방식으로 인덱스를 구축한다. 먼저, 2D-트리를 이용하여 전체 공간 차원을 유사한 수의 라인 세그먼트들이 배정되도록 다수의 셀들로 분할한다. 또한, 분할된 각 셀마다 시공간 차원 (x, y, t)에 대한 별도의 3차원 R*-트리를 두어 보다 상세한 인덱싱을 지원한다. 실험 결과에 의하면, 기존 기법에 비하여 작은 인덱스 구조를 갖으면서도 검색 성능면에서 300%~1000%까지의 성능 향상 효과를 갖는 것으로 나타났다.

1. 서론

이동 객체에 대한 사용자 질의는 크게 이동 객체의 미래 위치를 예측하여 검색하는 미래 예측 질의(future query)와 이동 객체의 과거에 움직인 위치를 검색하는 과거 이력 질의(historical query)로 구분된다. 미래 예측 질의의 경우 이동 객체의 현재 위치, 이동 속도, 이동 방향 등을 이용하여 미래 위치를 예측하는 것이 일반적이다[1, 2, 3]. 과거 이력 질의는 영역 질의(range query), 궤적 질의(trajectory query), 복합 질의(complex query)로 구분된다[4]. 영역 질의는 주어진 질의 영역 내에 존재하는 이동 객체를 검색하는 질의이며, 궤적 질의는 주어진 시간 간격 동안에 이동 객체가 움직인 경로를 검색하는 질의이다. 복합 질의는 영역 질의와 궤적 질의를 결합한 형태로 주어진 시간 간격 동안에 특정 영역에 있었던 이동 객체의 궤적을 검색하는 질의이다. 본 논문에서는 이들 중 영역 질의를 연구 대상으로 한다.

이동 객체의 궤적 정보는 다음과 같은 특징을 갖는다. (1) 시간의 흐름에 따라 공간적 위치가 지속적으로 변화하므로 갱신 비용이 크다. (2) 지속적으로 축적되어야 하므로 저장 공간 오버헤드가 매우 크다. (3) 축적된 대용량의 궤적 정보를 대상으로 하므로 검색 비용이 매우 크다. 이동 객체의 이러한 특징들을 고려하여 대용량 궤적 정보를 신속하게 저장, 관리, 검색할 수 있는 기술이 요구된다.

미래 예측 질의를 위한 인덱스 구조로는 VCI-트리[1], TPR-트리[2], TPR*-트리[3] 등이 제안된 바 있다. 이 중 가장 널리 이용되고 있는 TPR*-트리는 이동 객체들의 현 위치들에 대한 MBR(minimum bound rectangle)과 이동 속도를 함께 저장하는 CBR(conservative bounding rectangle)을 R*-트리[5] 내에 저장함으로써 이동 객체의 미래 위치를 빠르게 검색할 수 있도록 한다. 과거 이력 질의를 위한 인덱스 구조로는 3DR-트리[6], HR-트리[6], STR-트리[4], TB-트리[4], MV3R-트리[3], SETI[7] 등이 있다. 3DR-트리, HR-트리, SETI는 영역 질의, STR-트리, TB-트리, MV3R-트리는 궤적 질의를 주요 대상으로 한다. 본 연구의 대상인 영역 질의에서는 SETI가 인덱스 크기, 갱신 비용, 검색 성능 면에서 우수한 구조로 알려져 있다.

본 연구에서는 대용량 궤적 데이터베이스에서 영역 질의를 효과적으로 처리하기 위한 새로운 인덱싱 기법을 제안한다. 제안된 기법은 다음과 같은 이동 객체 데이터베이스의 두 가지 특성에 대한 관찰에 기반을 둔다. 첫째, 이동 객체의 궤적을 구성하는 라인 세그먼트들은 수집된 시간 순서대로 저장 및 인덱싱 된다. 따라서 저장되는 라인 세그먼트들의 시간 차원 t의 값은 단순히 증가되는 특성을 갖는다. 둘째, 영역 질의의 대상은 오래 전의 과거 세그먼트들 보다는 최근의 세그먼트들이 될 가능성이 높다.

제안된 기법에서는 우선 시간 차원을 다수의 시간 구간으로 분할하고, 인덱싱의 대상이 되는 전체 라인 세그먼트들을 구간별로 구분한다. 각 시간 구간에 속하는 라인 세그먼트들을 대상으로 별도의 인덱스를 사용한다. 최근 시간 구간과 대응되는 인덱스는 메인 메모리 내에서 관리되며, 나머지 시간 구간과 대응되는 인덱스들은 디스크 내에서 관리된다. 모든 라인 세그먼트의 삽입은 메인 메모리 내에서 관리되는 최근 시간 구간 인덱스 내에서 이루어지므로 빠른 처리가 가능하다. 또한, 최근 라인 세그먼트들에 대한 질의 특성상 빠른 질의 처리가 가능하게 된다.

2. 관련 연구

2.1. SETI의 특성

이동 객체의 궤적 T_i는 (moId, tId, <seg_1, seg_2, ..., seg_k>)로 구성된다. 여기서, moId는 이동 객체의 식별자이며, tId는 궤적 식별자이다. seg_j($0 < j \leq k$) 는 궤적 T_i을 구성하는 라인 세그먼트들이며, (sId, (xstart, ystart, tstart), (xend, yend, tend))로 구성된다. 여기서 sId는 세그먼트의 식별자이며, 해당 객체가 시간 구간 tstart와 tend에서 (xstart, ystart)으로부터 (xend, yend)로 이동하였음을 의미한다. 본 논문에서는 시간이 흐름에 따라 발생된 객체의 라인 세그먼트가 인덱스에 지속적으로 삽입된

다고 가정한다.

SETI[7]는 궤적의 시간 차원 값은 지속적으로 증가하는 반면, 공간 차원 값은 거의 변화하지 않는다는 특징을 이용하여 궤적들을 인덱싱한다. 먼저, 전체 공간 영역을 논리적인 셀(cell) 단위로 분할한 후, 각 셀 영역마다 희소 시간 인덱스(sparse time index)를 개별적으로 구성한다. 즉, 몇 개의 라인 세그먼트를 하나의 그룹으로 묶어서 데이터 페이지에 저장하고, 데이터 페이지 내의 라인 세그먼트들의 시간 차원 값들 중 최소 시간과 최대 시간을 희소 시간 인덱스의 엔트리로 저장한다. 희소 시간 인덱스 구조로는 1차원 R*-트리를 사용한다.

SETI는 이동 객체의 궤적에 추가되는 새로운 라인 세그먼트를 인덱스에 보다 빠르게 삽입하기 위하여 해쉬 테이블을 이용한다. 만약, 삽입하려는 이동 객체의 궤적 T_i의 위치가 A에서 A′로 변경되었다면 해쉬 테이블에서 A를 검색 한 후, A와 A′를 이용하여 새로운 라인 세그먼트를 구성한다. 다음 이 라인 세그먼트가 속하는 셀 영역을 찾은 후, 이와 대응되는 희소 시간 인덱스를 검색하여 해당 데이터 페이지에 이 라인 세그먼트를 삽입한다. 이때, 해쉬 테이블의 궤적 T_i의 위치는 A에서 A′으로 갱신된다. 그림 1은 SETI의 전체 구조를 나타낸 것이다.

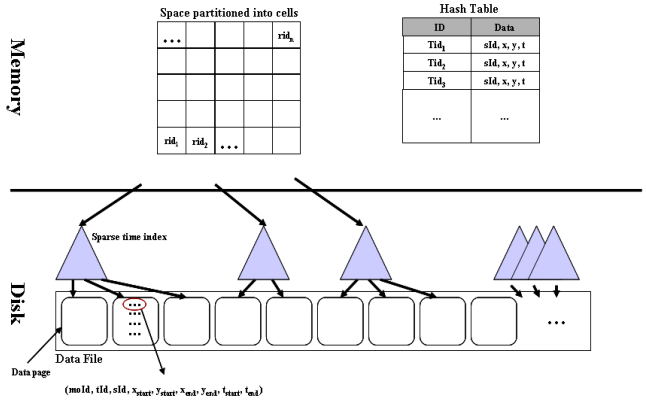

그림 1. SETI의 구조.

SETI에서의 검색 연산은 3단계로 나누어진다. 여과 단계(filtering step)에서는 주어진 질의 내 공간 조건을 만족하는 셀 영역을 검색한 후, 이와 대응되는 희소 시간 인덱스를 검색하여 질의 내 시간 조건을 만족할 가능성이 큰 후보 라인 세그먼트들이 저장되어 있는 데이터 페이지를 파악한다. 정제 단계(refinement step)에서는 여과 단계에서 얻어진 데이터 페이지를 액세스하여 후보 라인 세그먼트 중 질의내 시공간 조건을 모두 만족하는 라인 세그먼트들의 궤적 식별자를 얻는다. 제거 단계(elimination step)에서는 정제 단계에서 얻어진 궤적 식별자들 중 중복되는 식별자들을 제거하여 최종 결과를 얻는다.

2.2. SETI의 문제점

본 연구에서는 먼저 SETI의 문제점을 지적한다.

(1) 희소 시간 인덱스의 사용

희소 시간 인덱스를 사용하기 때문에 여과 단계에서는 엔트리가 가리키는 페이지 내의 라인 세그먼트들이 질의 내 시간 조건을 만족하는지의 여부를 파악할 수 없다. 따라서 정제 단계를 통하여 해당 데이터 페이지 내의 모든 후보 라인 세그먼트들을 대상으로 질의 내 시간 조건의 만족 여부를 따져야하는 오버헤드가 발생한다.

(2) 라인 세그먼트 집중

이동 객체들은 특정 영역에 집중되는 경향이 있으므로 라인 세그먼트들의 삽입 역시 이동 객체의 궤적이 특정 셀 영역에 집중되는 현상이 발생할 수 있다. 이로 인하여 라인 세그먼트들이 집중된 셀과 대응되는 희소 시간 인덱스에는 과부하가 걸릴 가능성이 크다. 이 결과, 일정한 검색 성능을 제공하기에는 어려움이 있다.

(3) 공간 인덱스의 미사용

분할된 셀 내부에 대하여는 별도의 공간 인덱스를 사용하지 않으므로 후보 라인 세그먼트의 수가 크게 증가하게 된다. 예를 들어, 주어진 질의 내 공간 조건 영역이 셀 영역의 공간 범위보다 매우 작은 경우, 희소 시간 인덱스에 대한 탐색 결과 질의 조건 영역 외부에 해당되는 다수의 라인 세그먼트들이 후보로 추천되며, 이 결과 검색 성능이 저하된다.

3. 제안하는 기법

3.1. 기본 개념

이동 객체의 궤적을 구성하는 라인 세그먼트들은 수집된 시간 순서대로 저장 장치에 저장 된다. 또한, 오래전의 과거 세그먼트보다는 현재 이동 중인 세그먼트나 최근에 이동이 완료된 세그먼트가 검색 대상이 될 가능성이 높다. 본 논문에서는 이동 객체의 이러한 특징을 이용하여 라인 세그먼트를 시간 흐름에 따라 효과적으로 삽입 및 검색할 수 있는 인덱싱 기법을 제안한다.

제안된 기법에서는 SETI와는 반대로 라인 세그먼트들을 우선 시간 차원으로 분할하고, 분할된 각 시간 구간에 속하는 라인 세그먼트들을 다시 공간 차원으로 분할하는 기법을 사용한다. 이와 같이, 최상단에서 시간 차원을 기반으로 분할하면 현재 이동 중인 이동 객체에 대한 라인 세그먼트의 삽입 연산은 마지막 시간 구간에서만 발생하게 된다. 또한 대부분의 인덱스는 디스크 내에서 관리하는 반면, 삽입 연산이 발생하는 마지막 시간 구간에 대한 인덱스는 메인 메모리 내에 상주시킨다. 이 결과 SETI에서와 같이 삽입 연산을 위하여 디스크 내의 희소 시간 인덱스에 대한 랜덤 액세스 문제를 해결할 수 있다.

삽입 연산을 담당하는 인덱스는 메인 메모리의 크기를 고려하여 미리 정해진 k개의 라인 세그먼트만을 삽입할 수 있도록 제한한다. 즉, 인덱스에 k번째의 라인 세그먼트에 대한 삽입이 이루어지면, 해당 인덱스를 메인 메모리에서 제거하고 디스크에 저장한다. 이후, 이 인덱스에 대하여는 검색 연산만이 발생하게 된다. 이 인덱스는 해당 시간 구간에 대하여 생성된 메인 메모리 내의 인덱스를 공간 차원으로 재분할하여, 분할된 m개의 공간 영역마다 서브 인덱스를 갖는 다단계 다중 인덱스 구조(mutilevel multiple indexes)이다. 제안된 기법에서는 분할된 각 셀이 $\lceil k/m \rceil$ 개의 라인 세그먼트를 저장하는 비 균일 셀 분할을 지원하며, 전체 공간 영역에 대한 분할 정보를 2D-트리[8]를 이용하여 관리한다. 이로써 SETI의 라인 세그먼트 집중, 공간 인덱스 미사용 등의 문제점을 해결할 수 있다. 또한, 제안된 기법에서는 검색 성능을 보다 향상시키기 위하여 분할된 공간 셀 영역 마다 작은 크기를 갖는 3차원 R*-트리를 두어 세부 인덱싱을 지원한다.

3.2. 인덱스 구조

그림 2는 제안하는 인덱스의 구조를 나타낸 것이다. 먼저, 시간 차원에 대한 분할 정보를 관리하는 시간 인덱스 테이블을 구성한다. 시간 인덱스 테이블은 n개의 엔트리로 구성되며, 각 엔트리는 대응되는 인덱스 I_i의 디스크 상의 위치를 저장한다. 시간 인덱스 테이블은 크기가 작으므로 메인 메모리에 상주시킬 수 있다. 이때, 각 인덱스 I_i는 미리 정해진 k개의 라인 세그먼트만을 저장한다.

여기서 주의할 점은 시간 구간 t_0, t_1, ..., t_{n-2}과 대응되는 인덱스 I_0, I_1, ..., I_{n-2}은 이동이 완료된 과거 궤적을 관리하기 위한 인덱스로 검색 연산만이 지원된다. 인덱스 I_i $(0 \le i \le n-2)$는 시간 구간 t_i를 2차원의 공간 차원을 기반으로 m개의 작은 셀 영역으로 재분할하여 저장한 인덱스로, 분할된 각 셀은 그림 2에서 나타난 바와 같이 $\lceil k/m \rceil$ 개의 라인 세그먼트를 저장하는 가변 크기를 갖는다. 인덱스 I_i는 해쉬 테이블이나 2D-트리 구조를 사용하여 구현될 수 있으며, 시공간 데이터 (x, y, t)에 대하여 해당 셀은

해당 영역 내에 존재하는 라인 세그먼트들을 대상으로 구축한 3차원 R*-트리 인덱스의 저장 위치를 엔트리로 갖는다. 또한, 3차원 R*-트리는 $\lceil k/m \rceil$ 개의 라인 세그먼트들에 대한 궤적 식별자와 세그먼트 식별자를 엔트리로 갖는다. 반면, 시간 구간 t_{n-1}과 t_n에 대응되는 인덱스 I_{n-1}과 I_n은 이동 중인 객체의 최근 세그먼트에 대한 삽입 연산과 검색 연산을 담당하며, 시공간 데이터 (x, y, t)에 대하여 구성된 3차원 R*-트리로 구성된다. 본 연구에서는 삽입 및 검색 연산의 효율성을 증가시키기 위하여 이들 인덱스 I_{n-1}과 I_n를 메인 메모리 내에 상주시킨다.

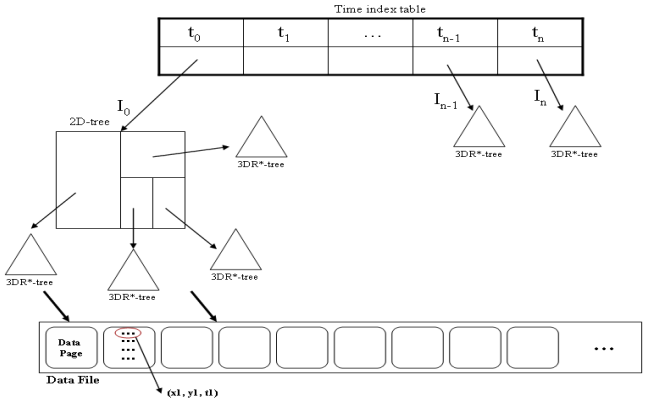

그림 2. 제안하는 인덱스 구조.

4. 성능 평가

4.1 실험 환경

실험에서는 시공간 데이터를 생성하는 GSTD(Generate SpatioTemporal Data)[9] 알고리즘의 시나리오 4(rectangles moving)에 의하여 생성된 [0, 1]사이에서 균일한 분포를 취하는 1,000개의 궤적 데이터를 사용한다. 각 궤적 데이터에 대하여 100만개(1M), 200만개(2M), 400만개(4M), 800만개(8M), 1,600만개(16M)의 세그먼트로 구성된 다섯 개의 실험 데이터 셋을 사용한다. 질의 영역으로는 실험 데이터로 사용된 궤적 데이터로부터 추출된 임의의 점에 대하여 0.01%, 0.1%, 1%의 영역 허용 범위를 갖는 질의 영역을 각각 1,000개씩 생성하여 사용한다. 각 실험 데이터와 질의 영역에 대하여 1,000개의 영역 질의를 수행한 평균 질의 처리 시간을 성능 평가 지수로 사용한다.

성능 평가는 다음 두 가지의 서로 다른 기법을 대상으로 한다. Ours는 본 논문에서 제안된 기법으로 라인 세그먼트들을 우선 시간 차원으로 분할하고, 분할된 각 시간 구간에 속하는 라인 세그먼트들을 다시 공간 차원으로 분할하는 방식이다. 성능 비교를 위한 기존의 기법으로서 SETI를 사용하며, 공간 차원에 대한 셀의 개수는 400(20×20)개이다. 이들 두 방식 모두 다차원 인덱스는 GIST[10]에서 제공하는 R*-트리를 사용하며, 데이터 페이지 크기로는 2KB를 사용한다.

본 실험을 위한 플랫폼으로는 Windows 2003 Server를 운영 체제로 사용하고, 3GB의 주기억 장치를 갖는 Intel Xeon 노코다 3.0GHz의 Intel Server를 사용한다.

4.2 실험의 기본 파라미터 값 설정

먼저, 시간 차원을 위한 k값과 공간 차원을 위한 d값을 결정한다. k는 하나의 시간 구간내에 저장되는 세그먼트의 개수이며, d는 하나의 셀내에 들어가는 세그먼트의 개수이다. 그림 3에 400만개의 세그먼트로 구성된 1,000개의 궤적 데이터에 대하여 k와 d값을 변화시키면서 질의 처리 시간을 측정한 결과를 보인다. 이때 질의 영역의 허용 범위는 0.1%이다. 실험 결과에 따르면, Ours는 대체적으로 k가 커질수록 질의 처리 시간이 증가하였다. 이는 k값의 증가는 하나의 시간 구간내에 속한 인덱스의 크기를 증가시키게 되므로 인덱스 검색 시간이 증가하기 때문이다.

또한, k값이 10,000개 이하에서는 d가 커질수록 질의 처리 시간이 감소하였다. 이는 d가 커지면 공간 차원에 해당하는 셀의 개수가 작아지고, 따라서 액세스되는 인덱스의 개수가 작아지기 때문이다. 그러나 k가 증가하는 경우, 인덱스의 개수는 작아지는 반면, 인덱스의 크기는 오히려 커지게 되므로 이러한 성능 향상을 보장할 수는 없다.

SETI는 k와 상관없이 거의 일정한 질의 처리 시간을 보였다. 이는 SETI는 공간 차원에 대하여 정적 분할 방식을 사용하므로 셀내에 저장되는 세그먼트의 개수 k는 설정하지 않기 때문이다. 이후 실험에서는 k와 d의 기본 값을 각각 1,000개와 500개로 설정하여 실험한다.

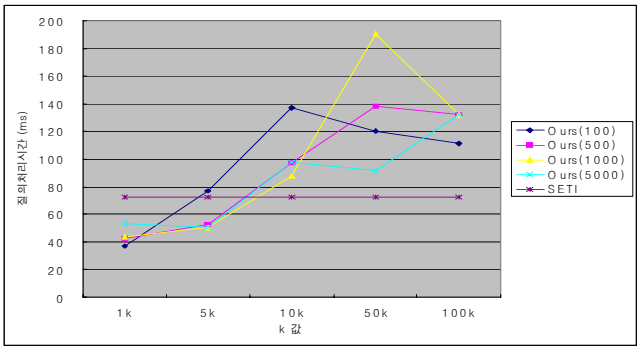

그림 3. k값의 변화에 따른 질의 처리 시간 비교.

4.3 실험 결과 및 분석

실험 1에서는 데이터의 크기 증가에 따르는 인덱스 크기를 비교한다. 인덱스의 크기는 인덱스 파일과 데이터 페이지 파일을 합한 크기이다. 그림 4의 실험 결과로부터, 두 방식 모두 데이터의 크기 증가에 따라 인덱스 크기가 증가함을 알 수 있다. 그러나 Ours는 SETI에 비하여 3배에서 4배까지의 저장 공간 감소 효과를 갖는 것으로 나타났다. 이는 SETI가 데이터 페이지에 (tId, sId, x_1, y_1, t_1, x_2, y_2, t_2) 구조의 아이템을 저장하는 것과 달리 Ours는 하나의 데이터 페이지에 동일 궤적의 세그먼트가 시간순으로 저장되므로 tId를 데이터 페이지의 헤더에 한번만 저장하고, (x, y, t) 구조의 아이템을 저장하기 때문이다.

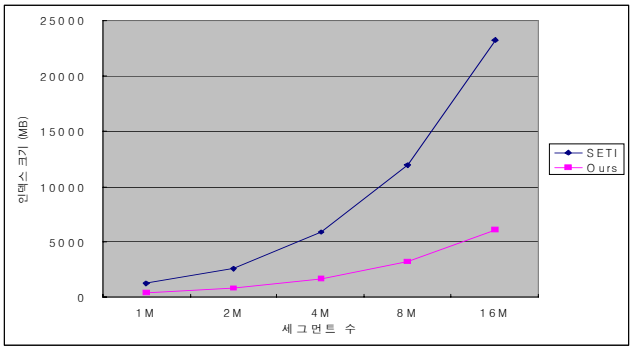

그림 4. 데이터 크기 변화에 따른 인덱스 크기 비교.

실험 2에서는 질의 처리 시간을 기준으로 각 기법의 성능을 비교한다. 그림 5는 데이터의 크기 증가에 따르는 각 기법의 질의 처리 시간의 변화를 나타낸다. 질의 영역 허용 범위는 0.1%로 고정하였다. 실험 결과로부터 데이터의 크기 증가에 따라 두 방식 모두 질의 처리 시간이 증가하지만, Ours가 SETI에 비하여 질의 처리 성능이 매우 우수함을 알 수 있다. 이는 Ours는 시간 차원 뿐만 아니라 공간 차원에 대한 인덱스를 사용하므로, SETI에 비하여 작은 크기의 인덱스를 사용하게 되며, 이 결과 인덱스 검색 시간이 감소하기 때문이다. 그림 5의 실험 결과로부

터 Ours는 SETI에 비하여 약 30%~100%까지의 성능 향상 효과를 가지는 것으로 나타났다.

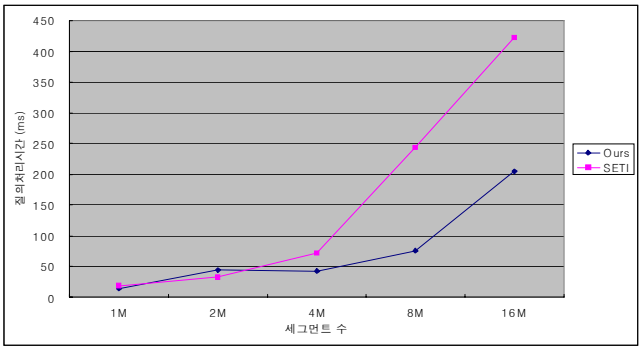

그림 5. 데이터 크기 변화에 따른 질의 처리 시간 비교.

그림 6은 질의 영역 허용 범위의 변화에 대한 각 방식의 질의 처리 시간의 변화를 나타낸다. 실험에는 세그먼트의 개수가 400만개(4M)인 데이터를 사용한다. 실험 결과로부터 질의 영역 허용 범위가 증가함에 따라 두 방식 모두 검색 대상이 되는 인덱스의 수가 증가하기 때문에 질의 처리 시간이 증가한다. 그러나 Ours가 SETI에 비하여 질의 처리 성능이 매우 우수함을 알 수 있다. 그 이유는 SETI는 희소 시간 인덱스를 사용하므로 질의 영역이 증가하게 되면 해당 데이터 페이지 내의 모든 후보 라인 세그먼트들을 대상으로 질의 내 시간 조건의 만족 여부를 판별하는 오버헤드가 급속하게 증가하기 때문이다. Ours는 SETI에 비하여 약 50%~120%까지의 성능 향상 효과를 가지는 것으로 나타났다.

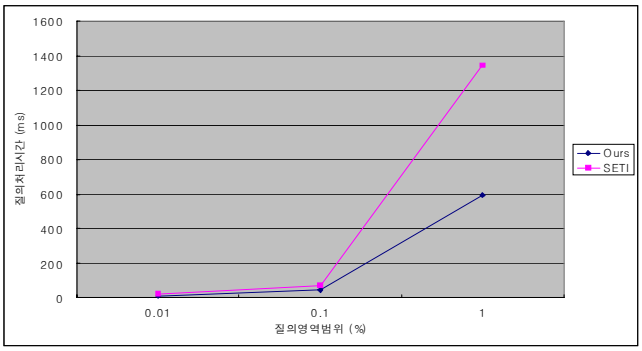

그림 6. 질의 영역 범위에 따른 질의 처리 시간 비교.

실험 3에서는 최근 질의에 대한 질의 처리 시간의 변화를 비교 및 평가한다. 실험에는 세그먼트 개수가 400만개(4M)인 궤적 데이터를 사용하며, 질의 영역 허용 범위는 0.1%로 설정한다. 최근 시간 구간내에 존재하는 궤적 데이터로부터 임의 추출하여 최근 질의를 생성한다.

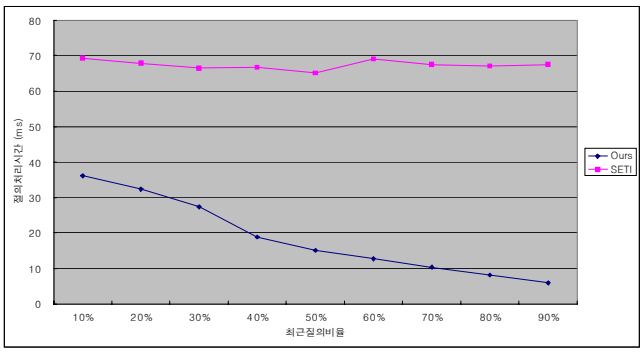

그림 7. 최근질의비율 변화에 따른 질의 처리 시간 비교.

그림 7의 실험 결과에 의하면, SETI는 최근 질의의 비율에 거의 영향을 받지 않고 일정한 질의 처리 시간을 보이는 반면, Ours는 최근 질의의 비율이 증가할수록 질의 처리 시간이 감소함을 알 수 있다. 그 이유는 SETI는 모든 세그먼트들을 디스크상에 인덱스로 구성하여 저장하는 반면, Ours는 최근에 삽입된 세그먼트들은 최근 시간 구간에 대한 인덱스로 구성하여 메인 메모리상에 관리하기 때문이다. Ours는 SETI에 비하여 100%~1000%까지의 성능 개선 효과를 갖는 것으로 나타났다.

5. 결론

본 논문에서는 대용량 궤적 데이터베이스를 위한 효율적인 인덱스 구조와 이를 기반으로 하는 영역 질의 알고리즘을 제안하였다. 제안된 기법에서는 이동 객체의 궤적에 속하는 라인 세그먼트들이 시간 흐름에 따라 순서적으로 삽입되며, 저장이 오래된 과거 궤적보다는 최근 궤적이 질의의 검색 대상이 될 가능성이 높다는 특징을 이용하여 삽입과 검색 효율성을 극대화시킬 수 있도록 다음과 같은 전략을 사용한다. 먼저, 라인 세그먼트들을 시간 구간별로 구분함으로써 현재와 과거 라인 세그먼트들을 분리한다. 높은 액세스가 빈번하게 발생하는 최근 라인 세그먼트들에 대하여는 메인 메모리상에 인덱스를 관리하며, 대부분의 과거 라인 세그먼트들에 대해서는 디스크 상에 인덱스들을 관리한다. 실험 결과에 의하면, 본 논문에서 제안한 다단계 다중 인덱스 기법이 기존의 SETI를 이용한 기법에 비해 작은 크기의 인덱스 구조를 사용하면서도, 검색 연산면에서 최대 1000%까지의 성능 향상되었다.

Acknowledgement

본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT 연구센터 지원사업의 연구결과로 수행되었음
(IITA-2008-C1090-0801-0040)

참고 문헌

[1] M. A. Nascimento and J. R. O. Silva, "Towards Historical R-trees," In Proc. ACM Symposium on Applied Computing, pp. 235-240, 1998.

[2] S. Saltenis, C. S. Jensen, and S. T. Leutenegger, "Indexing the Positions of Continuously Moving Objects," In Proceedings of the 2000 ACM-SIGMOD Conference, pp. 331-342, 2000.

[3] Y. Tao and D. Papadias, "MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries," In Proc. VLDB Conference, pp. 431-440, 2001.

[4] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel Approaches in Query Processing for Moving Object Trajectories," In Proceedings of the 26st VLDB Conference, pp. 395-406, 2000.

[5] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles," In Proceedings of the 1990 ACM-SIGMOD Conference, pp. 322-331, 1990.

[6] Y. Theodoridis, and T. K. Sellis, "Spatio-Temporal Indexing for Large Multimedia Applications," In International Conference on Multimedia Computing and Systems, pp. 441-448, 1996.

[7] V. P. Chakka, A. C. Everspaugh, J. M. Patel, "Indexing Large Trajectory Data Sets With SETI," In Proceedings of the 2003 CIDR Conference, 2003.

[8] J.L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," Comm. of the ACM, 18(9), 1975.

[9] Y. Theodoridis, J. R. O. Silva and M. A. Nascimento, "On the Generation of Spatiotemporal Datasets," In Proc. Int'l Symp. on Large Spatial Databases, SSD, pp. 147-164, 1999.

[10] Berkeley University, The GiST Indexing Project, http://gist.cs.berkely.edu, 2007.