미래 위치 검색을 위한 이동 객체의 효과적인 갱신 기법

[1]복경수[O] [1]김명호 [2]조기형 [2]유재수
[1]한국과학기술원 전산학과
[2]충북대학교 정보통신공학과

{ksbok[O], mhkim}@dbserver.kaist.ac.kr
{yjs,khjoe}@cbucc.chungbuk.ac.kr

Efficient Update Strategies of Continuously Moving Objects for Retrieving Future Positions

[1]KyoungSoo Bok[O] [1]MyoungHo Kim [2]KiHyung Cho [2]JaeSoo Yoo
[1]Dept. of Computer Science, Korea Advanced Institute of Science and Technology
[2]Dept. of Computer and Communication Engineering, Chungbuk National University

요 약

1990년대 중반 이후 위치 기반 서비스에 대한 활용이 증가되면서 이동 객체를 효과적으로 저장, 관리하기 위한 많은 연구들이 진행되고 있다. 본 논문에서는 미래 위치를 검색하기 위한 시공간 색인 구조에서 계속적인 이동 객체의 위치 변화를 효과적인 갱신하기 위한 기법을 제안한다. 제안하는 갱신 기법은 이동 객체의 위치 변화에 따른 색인 구조의 재구성 시간을 감소시키기 위해 전통적인 공간 분할 방식의 색인 구조를 변형한다. 이동 객체의 위치 변화를 처리하기 위해 보조 색인 구조를 통해 단말 노드를 직접 접근하여 상향식으로 갱신을 수행한다.

1. 서 론

위치 기반 서비스에 대한 활용이 증가되면서 시간의 변화에 따라 연속적인 위치를 변화하는 이동 객체를 빠르게 색인하기 위한 연구들이 진행되고 있다[1, 2]. 기존에 이동 객체의 미래 위치를 검색하기 위해 제안된 색인 구조들은 중간 노드에 자식 노드에 포함된 이동 객체를 포함하는 영역과 미래 위치를 검색하기 위한 파라미터를 저장한다[2, 3, 4].

기존에 제안된 R-트리 기반의 색인 구조들은 이동 객체의 위치 변화를 갱신하기 위해 색인 구조 전체를 순회하면서 이동 객체의 이전 위치를 삭제하고 새로운 위치를 삽입하는 과정을 수행한다. 이러한 갱신 과정은 추가적인 연산을 필요로 하며 이와 함께 많은 I/O를 발생시켜 색인 구조의 성능을 저하시키는 문제가 발생한다[5]. 또한, 계속적인 갱신 과정에서 분할된 영역들 사이에 선별력이 저하되어 검색 성능이 저하되는 문제점이 있다.

본 논문에서는 기존 R-트리 기반 색인 구조의 문제점을 해결하기 위한 새로운 갱신 기법을 제안한다. 제안하는 갱신 기법에서 사용하는 색인 구조는 공간 분할 방식 색인 구조를 변형한 색인 구조와 해쉬 테이블로 구성된 보조 색인 구조를 통해 색인을 구성한다. 제안하는 갱신 기법은 색인 구조의 단말 노드를 직접 접근하고 이동 객체의 갱신으로 노드의 재구성이 필요할 경우 각 노드에 존재하는 부모 노드에 대한 포인터를 이용하여 상향식으로 색인을 구성한다. 또한, 이동 객체의 삭제 또는 오버플로우를 처리하기 위해 소요되는 시간을 감소시키기 위한 새로운 삭제 기법과 오버플로우 처리 기법을 제안한다.

본 논문의 나머지 부분은 다음과 같이 구성한다. 2장에서는 기존 색인 구조에 대해 기술하고 3장에서는 본 논문에서 제안하는 갱신 기법과 색인 구조에 대해 기술한다. 4장에서는 성능 평가를 통해 제안하는 갱신 기법의 우수성을 입증하고 5장에서는 논문의 결론 및 향후 연구에 대해 기술한다.

본 과제(결과물)는 정보통신부의 IT기초기술연구지원사업(정보통신연구진흥원)으로 수행한 연구결과물 입니다.

2. 관련 연구

이동 객체의 미래 위치를 검색하기 위해 S. Prabhakar는 이동 객체의 변화에 따른 색인 구조에 대한 갱신 비용을 줄이기 위해 자식 노드에 포함된 이동 객체의 최대 속도를 이용하여 색인을 구성하는 VCI-트리를 제안하였다[3]. VCI-트리는 기존의 R-트리 기반 색인 구조에 v_{max} 라는 값을 부가적으로 사용하여 이동 객체를 색인한다. VCI-트리는 색인을 구성하는 시점에서는 모든 객체에 대한 위치를 정확하게 유지하지만 객체들이 임의 방향으로 이동할 수 있기 때문에 특정 시점에서는 정확한 위치를 유지하지 못한다.

S. Saltenis는 이동 객체의 미래 위치를 검색하기 해 기존 R*-트리 기반의 색인 구조를 변형한 TPR-트리를 제안하였다[2]. TPR-트리의 단말 노드에는 이동 객체의 위치를 저장하고 중간 노드에는 자식 노드에 포함된 모든 이동 객체를 포함하는 영역과 속도 정보를 표현한다. TPR-트리는 이동 객체의 계속적인 갱신이 발생할 경우 많은 비용을 소요하며 기존 MBR 영역의 외부에 이동 객체가 삽입될 경우 영역을 재구성하기 위해 많은 시간을 소요한다.

Y. Tao는 검색 과정에서 접근하는 노드의 수를 감소시키기 위해 기존 TPR-트리의 삽입과 삭제 알고리즘을 개선한 TPR*-트리를 제안하였다[4]. TPR*-트리는 새로운 이동 객체를 삽입할 위치를 판별하기 위해 큐를 이용하여 루트 노드에서부터 자식 노드를 삽입하기에 적절한 노드의 우선 순위에 따라 노드를 기록하여 새로운 이동 객체를 삽입할 자식 노드를 판별한다.

3. 제안하는 갱신 기법

3.1 색인 구조의 특징

이동 객체의 계속적인 위치 변화에 대한 갱신을 효과적으로 수행하기 위해 본 논문에서는 공간 분할 방식의 색인 구조를 이용하여 색인을 구성한다. 그림 1은 제안하는 갱신 기법에서

사용되는 색인 구조를 나타낸 것이다. 그림 1에서 s_1는 공간 분할을 수행한 위치이고 분할 영역에 표시된 검은색의 화살표는 속도 정보를 나타낸다. 제안하는 색인 구조는 공간 분할 방식에 의해 분할된 주 색인 구조와 이동 객체가 저장된 단말 노드를 접근하기 위한 보조 색인 구조로 구성된다.

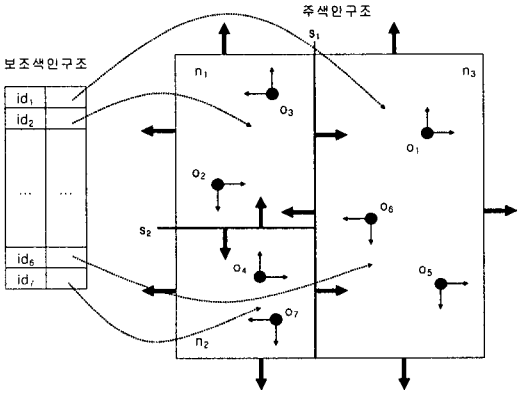

그림 2 색인 구조

주 색인 구조는 현재 및 미래 위치를 검색하기 위해 실제 이동 객체의 위치를 저장한다. 주 색인 구조는 이동 객체의 계속적인 위치 변화에 따른 노드의 재구성 시간을 감소시키기 위해 공간 분할 방식 색인 구조를 변형한 높이 균형 트리이다. 주 색인 구조의 중간 노드에는 자식 노드에 존재하는 이동 객체를 포함하는 영역과 함께 이동 객체의 위치 위치를 검색하기 위한 부가적인 정보를 표현한다. 또한, 각 노드의 헤더에는 부모 노드를 직접 접근하기 위한 포인터를 저장하여 상향식으로 갱신을 수행할 수 있다.

보조 색인 구조는 이동 객체가 최신 위치를 저장하고 있는 주 색인 구조의 단말 노드를 직접 접근하기 위한 해쉬 테이블로 구성되어 있다. 해쉬 테이블은 단말 노드에 저장되어 있는 이동 객체를 직접 접근하기 위해 $<id, ptr>$로 구성된다. 이때, id는 단말 노드에 저장된 이동 객체에 대한 식별자, ptr는 이동 객체가 저장되어 있는 단말 노드에 대한 포인터이다.

3.2 삽입

본 논문에서는 공간 분할 방식을 이용하여 색인을 구성하기 때문에 분할된 영역들 사이에 겹침이 발생하지 않으며 주 색인 구조의 중간 노드에는 자식 노드에 대한 속도 정보를 저장하기 때문에 이동 객체의 미래 위치를 검색할 수 있다. 삽입은 기존에 제안된 공간 분할 방식과 유사하게 수행한다. 이동 객체를 삽입하기 위해서는 이전에 삽입된 이동 객체에 대한 갱신인지 새로운 객체의 삽입인지를 판별하기 위해 해쉬 테이블을 검색한다. 만약 이전에 삽입된 이동 객체의 위치에 대한 갱신을 수행하는 것이라면 갱신을 수행하고 새로운 이동 객체에 대한 삽입이라면 객체를 삽입하기 위한 단말 노드를 탐색한다.

이동 객체의 삽입으로 노드에 오버플로우가 발생하면 직접 분할을 수행하는 것이 아니라 오버플로우가 발생한 형제 노드를 검색하여 병합 분할(merge and split)을 수행한다. 병합 분할은 오버플로우가 발생한 노드와 형제 노드를 병합하고 오버플로우가 발생한 노드에 존재하는 일부 객체들을 형제 노드에 삽입한다. 만약 병합 분할이 가능한 형제 노드가 존재하지 않는 경우에는 객체의 이동성을 고려하여 분할을 수행한다. 예를 들어, 2차원 데이터 공간에 새로운 객체 e가 삽입되어 노드에 오버플로우가 발생할 경우 n_1 노드 내에 존재하는 o_3을 n_2에 삽입하고 기존 분할 위치 s_2는 s_3로 변경하여 오버플로우를 처리한다.

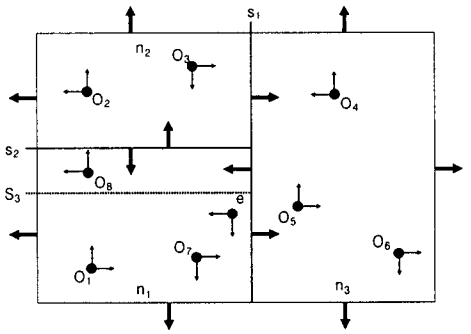

그림 2 병합 분할 수행

현재 서비스를 수행 중인 이동 객체의 위치는 주기적으로 서버에 전송된다. 특정 시간 동안 서버로 위치를 전송하지 않는 객체는 장애가 발생하거나 서비스를 중지한 것으로 판별한다. 서비스를 중지한 이동 객체를 삭제하기 위해서 본 논문에서는 이동 객체의 삽입이나 갱신을 수행하기 위해 단말 노드를 접근하면 서비스를 중지한 이동 객체를 판별하여 갱신을 수행한다.

3.3 갱신

본 절에서는 이동 객체의 위치 변화에 대한 갱신을 효과적으로 처리하기 위해 부분 상향식 갱신(PBU : Partial Bottom-up Update) 기법과 완전 상향식 갱신(FBU : Full Bottom-up Update) 기법을 제안한다.

3.3.1 부분 상향식 갱신 기법

부분 상향식 갱신 기법은 이동 객체의 새로운 위치를 갱신하기 위해 접근한 단말 노드의 영역 내에 이동 객체의 새로운 위치가 존재하는 경우에는 이동 객체의 새로운 위치를 삭제하고 새로운 위치를 단말 노드에 삽입한다. 만약 이동 객체의 갱신으로 미래 위치를 검색하기 위한 정보가 변경될 경우 색인 구조를 전체를 순회하지 않고 부모 노드에 대한 포인터를 이용하여 상향식으로 변경된 내용을 반영한다. 이에 반해, 이동 객체의 새로운 위치가 이동 객체의 이전 위치가 존재하는 단말 노드의 영역 내에 존재하지 않는 경우에는 단말 노드에 존재하는 이동 객체의 위치를 삭제하고 상향식으로 갱신을 수행한다. 즉, 주 색인 구조 전체를 순회하면서 이동 객체의 새로운 위치를 삽입하기 위한 단말 노드를 검색하고 새로운 위치를 삽입한다.

3.3.2 완전 상향식 갱신 기법

공간 분할 방식의 색인 구조에서는 이동 객체의 갱신되는 위치는 대부분 기존 단말 노드에 존재하거나 이웃한 형제 노드 내에 존재한다. 따라서, 색인 구조 전체를 순회하지 않고 현재 갱신을 수행하는 부모 노드를 접근하면 이동 객체의 새로운 위치를 삽입하기 적절한 노드를 검색할 가능성이 매우 높다. 완전 상향식 갱신 기법은 이동 객체의 새로운 위치가 기존 단말 노드에 존재하지 않는 경우에도 상향식 갱신을 수행한다.

그림 3은 완전 상향식 갱신을 수행하는 과정을 나타낸 것이다. 그림 3에서 각 번호 ①는 완전 상향식 갱신을 처리하는 순서를 나타낸 것이다. 완전 상향식 갱신 기법에서는 o_7의 새로운 위치 o_7'가 기존 단말 노드에 존재하지 않는 경우 부모 노드를 계속 접근하여 o_7'의 포함하는 노드를 판별한다. 이때, 서

비스를 중지한 객체들을 삭제하여 노드의 정보가 변경되면 접근한 부모 노드에 이를 함께 반영한다.

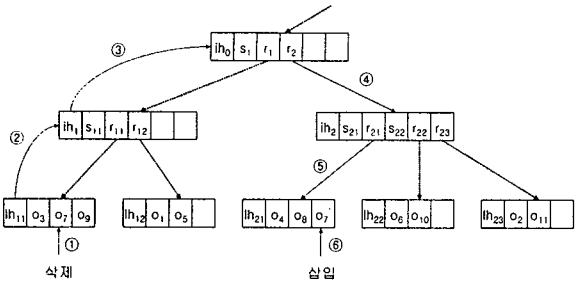

그림 4 상향식 갱신 과정

4. 실험 및 성능 평가

성능 평가를 위해 제안하는 색인 구조와 TPR-트리를 펜티엄 IV 2.8GHz 프로세서에 256Mbyte의 메모리를 가지는 시스템에서 구현한다. 구현한 색인 구조에서 노드의 크기는 4kbyte이며 C언어를 통해 구현한다. 성능평가에 사용된 데이터 집합은 GSTD[6]을 통해 2차원 공간에서 균등(U), 가우시안(G) 그리고 스큐(S) 분포로 존재하는 100,000개 이동 객체를 생성한다. 편의상 제안하는 색인 구조는 PIS-트리(Proposed Index Structure-tree)라 한다.

4.1 삽입 성능

이동 객체에 대한 삽입 성능을 평가하기 위해 10,000개에서 100,000개의 이동 객체를 삽입하는 시간을 분석한다. 그림14에서 그림 16은 이동 객체에 대한 삽입 시간을 나타낸 것이다. 기존에 제안된 TPR-트리는 이동 객체의 수가 증가할수록 삽입 시간이 급격히 증가되지만 제안하는 색인 구조는 삽입되는 이동 객체의 수가 증가되어도 삽입 시간에 많은 영향을 받지 않는다. 제안하는 PIS-트리는 기존에 제안된 TPR-트리에 비해 450%의 성능 향상을 보인다.

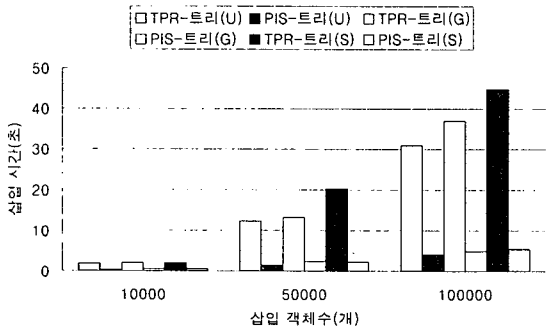

그림 5 삽입 성능

4.2 갱신 성능

이동 객체의 갱신 기법을 비교하기 위해 100,000개의 이동 객체의 삽입하고 1,000개에서 10,000개의 이동 객체를 갱신하기 위해 소요되는 시간을 비교 분석한다. 그림 5는 이동 객체의 분포 특성에 따른 갱신 시간을 비교한 것이다. 기존에 제안된 색인 구조는 갱신되는 이동 객체의 수가 증가될수록 갱신 시간이 급격히 증가된다. 이에 반해, 제안하는 색인 구조는 갱신되는 이동 객체의 수가 증가되어도 갱신 시간에 많은 영향을 받지 않는다. 제안하는 PIS-트리의 완전 상향식 갱신 기법은 기존에 제안된 TPR-트리에 비해 약 340%의 성능 향상을 보인다.

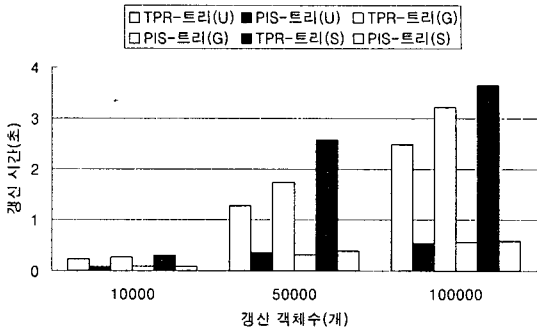

그림 6 갱신 성능

5. 결론 및 향후 연구

본 논문에서는 이동 객체의 미래 위치 검색을 위한 색인구조에 대한 갱신 기법을 제안하였다. 제안하는 갱신 기법에서 사용하는 색인구조는 공간 분할 방식으로 구성하며 계속적인 위치 변화에 대한 갱신을 효과적으로 수행하기 위해 보조 색인 구조를 이용하여 이동 객체의 이전 위치가 저장되어 있는 노드를 직접 접근할 수 있도록 한다. 각 노드에는 부모 노드를 직접 접근하기 위한 포인터를 유지하고 이동 객체의 갱신으로 노드의 변경이 발생할 경우 색인 구조 전체를 순회하지 않고 상향식으로 갱신을 수행한다.

향후 연구 방향으로 연속 질의 처리 기법에 대한 연구를 수행할 예정이다.

참고 문헌

[1] M. F. Mokbel, T. M. Ghanem and W. G. Aref, "Spatio-Temporal Access Methods", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol.26, No.2, pp.40-49, 2003.

[2] S. Saltenis, C. S. Jensen, S. T. Leutenegger and M. A. Lopez, "Indexing the Positions of Continuously Moving Objects", Proc. the 2000 ACM SIGMOD International Conference on Management of Data, pp.331-342, 2000.

[3] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref and S. E. Hambrusch, "Query Indexing and Velocity Constrained Indexing : Scalable Techniques for Continuous Queries on Moving Objects", IEEE Transactions on Computers, Vol.51, No.10, pp.1124-1140, 2002.

[4] Y. Tao, D. Papadias and J. Sun, "The TPR*-Tree : An Optimized Spatio-Temporal Access Method for Predictive Queries", Proc. the 29th International Conference on Very Large Data Bases, pp. 790-801, 2003.

[5] B. C. Ooi, K. L. Tan and C. Yu, "Frequent Update and Efficient Retrieval: an Oxymoron on Moving Object Indexes?", Proc. the 3rd International Conference on Web Information Systems Engineering Workshops, pp.3-12, 2002.

[6] Y. Theodoridis, R. Silva and M. Nascimento, "On the Generation of Spatiotemporal Datasets", Proc. the 6th International Symposium on Spatial Databases, pp.147-164. 1999.