대용량 이동객체의 위치정보 관리를 위한 S-GRID를 이용한 분산 그리드 기법

(Distributed Grid Scheme using S-GRID for Location Information Management of a Large Number of Moving Objects)

김 영 창* 김 영 진** 장 재 우***
(Youngchang Kim) (Youngjin Kim) (Jaewoo Chang)

요 약 최근 모바일 기기 및 무선 통신의 발달로 인하여 다양한 위치 기반 서비스에 대한 연구가 증대되고 있으며, 이러한 위치 기반 서비스의 대표적 질의인 k-최근접 질의를 효율적으로 처리하기 위한 연구가 활발히 수행되어 왔다. 기존 연구들은 질의 처리 성능의 향상을 위해, 공간 네트워크 상의 POI와 노드 사이의 거리를 미리 계산하는 pre-computation 기법을 사용한다. 그러나 이러한 pre-computation 기법들은 검색 대상이 되는 POI의 변경을 효과적으로 처리하지 못하는 단점을 갖는다. 본 논문에서는 기존 pre-computation 기법들의 단점을 극복하고, 대용량 이동객체의 위치정보를 효율적으로 관리하기 위하여 S-GRID를 이용한 분산 그리드 기법을 제안한다. 아울러 제안하는 분산 그리드 기법을 위한 k-최근접 질의 처리 알고리즘을 제시한다. 마지막으로, S-GRID 및 분산 그리드 기법의 k-최근접 질의처리 알고리즘의 성능 평가를 통해, 제안하는 기법의 우수성을 입증한다.

키워드 : 분산 그리드 기법, k-최근접 질의 처리 알고리즘, 공간 네트워크

Abstract Recently, advances in mobile devices and wireless communication technologies require research on various location-based services. As a result, many studies on processing k-nearest neighbor query, which is most important one in location-based services, have been done. Most of existing studies use pre-computation technique to improve retrieval performance by computing network distance between POIs and nodes beforehand in spatial networks. However, they have a drawback that they can not deal with effectively the update of POIs to be searched. In this paper, we propose a distributed grid scheme using S-GRID to overcome the disadvantage of the existing work as well as to manage the location information of a large number of moving objects in efficient way. In addition, we describe a k-nearest neighbor(k-NN) query processing algorithm for the proposed distributed grid scheme. Finally, we show the efficiency of our distributed grid scheme by making a performance comparison between the k-NN query processing algorithm of our scheme and that of S-GRID.

Key words : Distributed Grid scheme, k-nearest neighbor query processing algorithm, Spatial network

1. 서론

최근 PDA, 휴대폰, GPS와 같은 모바일 기기 및 무선 통신의 발달로 인하여 위치 기반 서비스의 요구가 증대되고 있다. 위치 기반 서비스의 주요 대상은 사용자, 차량 등과 같은 이동객체이며, 이들은 이상적인 유클리디언(Euclidean) 공간 대신 도로나 철도와 같은 공간 네트워크(spatial network) 상에서 이동한다. 이러한 위치 기반 서비스에서 대표적인 질의로는 범위 질의, k-최근접 질의, closest pair, e-distance 조인 질의 등이 있다. 이중 가장 중요한 질의는 이동객체로부터 가장 가까운 k 개의 POI(point of interest) 및 이동객체를 검색하는 k-최근접 질의이며, 이를 위해 많은 연구들이 수행되었다[1, 2, 3, 4]. 기존 연구들은 검색 성능을 향상시키기 위해, 공간 네트워크 상의 노드와 POI 사이의 거리를 미리 계산하는 pre-computation 기법을 사용한다. 그러나 pre-computation 기법들은 위치정보가 빈번하게 갱신되는 이동객체를 효과적으로 처리하니 못하는 단점을 지닌다. 이

† 본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신 인력양성사업으로 수행된 연구 결과임.

* 전북대학교 전자정보공학부 컴퓨터공학 박사과정, yckim@dblab.chonbuk.ac.kr
** 전북대학교 전자정보공학부 컴퓨터공학 박사과정, yzkim@chonbuk.ac.kr
*** 전북대학교 전자정보공학부 컴퓨터공학 교수, jwchang@chonbuk.ac.kr(교신저자)

논문접수 : 2008.07.10 수정일 : 1차 2008.08.28 심사완료 : 2008.09.12

러한 단점을 극복하기 위하여, 최근 공간 네트워크 정보를 POI 정보와 독립적으로 관리하기 위한 S-GRID (scalable grid)가 제안되었다[5]. S-GRID 는 공간 네트워크를 일정한 크기의 2차원 그리드 셀로 나누고 각 그리드 셀 영역에 포함되는 공간 네트워크 정보만을 미리 계산하여 관리한다. 아울러 검색 대상이 되는 POI나 이동객체는 기존 R 트리를 사용하여 전체적인 성능을 향상시켰다.

한편, 모바일 기기 및 무선 통신의 발달로 인한 이동 단말기의 보급 확대로 인하여, 수집 가능한 이동객체 데이터가 급증하였다. 자동차의 경우 국내에만 약 1,500만 대 이상이 운행 중인 것으로 집계되었으며[6], 이러한 대용량의 이동 객체의 위치정보를 효율적으로 관리하기 위해서는 분산 처리 연구가 필수적이다. 이를 위해 대용량 이동객체를 위한 기존 분산 처리 연구가 수행되었으나[7, 8], 이들은 유클리디언 거리를 기반으로 하기 때문에 공간 네트워크상의 이동객체를 지원하기 어렵다는 단점이 있다. 따라서 본 논문에서는 공간 네트워크상에서 대용량의 이동 객체 데이터를 효율적으로 지원하기 위하여 S-GRID를 이용한 분산 그리드 기법을 설계한다. 아울러 제안하는 분산 그리드 기법을 위한 k-최근접 질의처리 알고리즘을 제안하며, 기존 S-GRID와 제안하는 기법의 성능 비교를 수행한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구인 S-GRID를 소개하고, 3장에서는 S-GRID를 이용한 분산 그리드 기법을 설계한다. 4장에서는 제안하는 기법과 S-GRID의 성능 평가에 대해서 기술하며, 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. S-GRID

S-GRID는 기존 pre-computation 기법들의 단점인 공간 네트워크 정보의 갱신 및 검색의 대상이 되는 POI 및 이동객체의 위치정보 갱신으로 인한 검색성능 저하를 해결하기 위하여 제안되었다[5]. S-GRID는 공간 네트워크를 2차원의 고정 크기의 그리드 셀로 나누어 저장한다. 각 그리드 셀은 그리드 셀과 공간 네트워크의 교차점(intersection point)을 포함하여 노드 사이의 거리를 미리 계산하여 저장한다. 저장된 정보는 크게 Vertex-Edge, Vertex-Border, Cell-Border의 3개의 컴포넌트로 구성된다. 첫째, Vertex-Edge 컴포넌트는 각 노드(vertex)와 인접한 노드 사이의 거리를 미리 계산하여 저장한다. 둘째, Vertex-Border 컴포넌트는 각 노드로부터 해당 셀의 모든 경계점(border point)까지의 거리를 저장하며, 마지막으로 Cell-Border 컴포넌트는 셀의 각 경계점 사이의 거리를 계산하여 저장한다. 아울러 임의의 셀에 대한 빠른 접근을 위해 해당 셀이 저장된 페이지 정보를 갖는 해시 테이블을 메모리에 유지한다. 그림 1 은 S-GRID의 전체적인 구조를 나타낸다. S-GRID의 k-최근접 질의 처리 알고리즘은 두 개의 우선순위 큐 Qdp, Qv를 이용하며, 내부 확장과 외부 확장으로 구성된다. Qdp에는 검색된 POI를 질의점까지의 거리순서로 저장하며, Qv는 확장할 노드를 질의점까지의 거리 순서로 저장한다. 이때, 확장할 노드가 경계점이면 외부 확장을 그렇지 않으면 내부 확장을 수행한다. 먼저 내부 확장은 Vertex-Edge 컴포넌트와 Vertex-Border 컴포넌트 정보를 이용하여 질의점이 속한 셀 내에서 이루어지며, 확장 방법은 기존 INE[1] 방법과 동일하다. 둘째, 외부 확장은 경계점을 공유하는 이웃 셀의 Cell-Border 컴포넌트 정보를 검색하여 질의점으로부터 이웃 셀의 모든 경계점까지의 거리를 계산하여 Qv에 삽입한다. 또한, Vertex-Border 컴포넌트 정보를 검색하여 질의점으로부터 해당 셀에 존재하는 모든 POI까지의 거리를 계산하여 Qdp에 삽입한다. 마지막으로 질의 처리 알고리즘은 질의점으로부터 확장할 노드까지의 거리가 검색된 k 번째 POI까지의 거리보다 크면 종료한다.

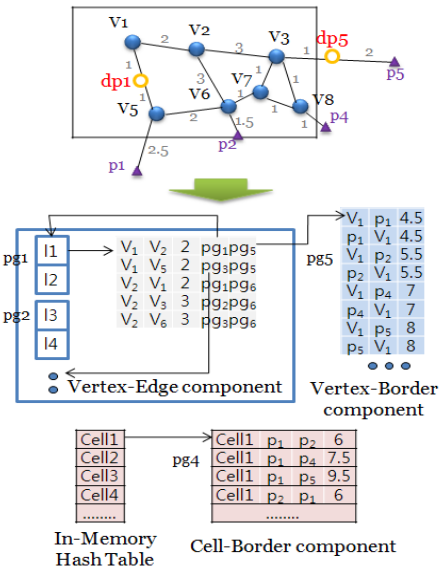


그림 1. S-GRID의 전체적인 구조

3. S-GRID를 이용한 분산 그리드 기법의 설계

본 절에서는 공간 네트워크를 위한 분산 그리드 기법을 설계하고, 아울러 분산 그리드 기법을 위한 k-최근접 질의 처리 알고리즘을 제안한다.

3.1 분산 그리드 기법의 전체구조

S-GRID는 공간 네트워크 정보를 2차원의 고정된 크기의 그리드 셀 단위로 나누어 저장하기 때문에 셀 단위의 분산 처리가 용이하다. 따라서 이동객체의 위치정보를 기반으로 해당하는 그리드 셀에 매핑 시킨 후 셀별로 R 트리를 이용하여 이동객체를 위한 색인구조의 구성이 가능하다. 이를 이용하여 분산 그리드 기법을 설계한다. 제안하는 분산 그리드 기법의 전체적인 구조는 <그림 2>와 같다. 전체 공간 네트워크는 고정 크기의 n*n 그리드

셀로 나누어진다. 각 그리드 셀은 In-Memory 셀 테이블, Vertex-Edge, Vertex-Border, Cell-Border의 4개 컴포넌트로 구성되며, 하나의 그리드 셀 정보는 하나의 서버에서 관리한다. 그리드 셀의 수보다 서버의 수가 적을 경우에는 생성된 그리드 셀을 각 서버로 균등 분할하여 할당하고, 서버에 할당된 셀의 수만큼 스레드를 생성하여 각 스레드가 가상의 서버역할을 수행한다. 이때, Vertex-Edge, Vertex-Border, Cell-Border 컴포넌트는 기존 S-GRID와 동일한 구조를 갖는다. S-GRID 에서는 다른 셀의 Cell-Border 컴포넌트에 빠르게 접근하기 위해 In-Memory 해시 테이블을 유지하며, 한편 분산 그리드 기법에서는 다른 셀의 Cell-Border 컴포넌트를 유지하지 않기 때문에, In-Memory 해시 테이블을 필요로 하지 않는다. 그러나 빠른 질의 처리를 위해 각 셀에 존재하는 POI의 수를 저장하는 셀 테이블을 메모리상에 유지한다. 분산 그리드 기법에서는 질의점으로부터 가장 가까운 k 개의 POI를 검색하기 위하여, 자신의 셀과 이웃 셀을 관리하는 서버로부터 필요한 정보를 검색한다. 이를 위해 질의를 처리하는 서버는 다른 서버로 질의를 전달하여 이웃 셀에 존재하는 POI를 검색한다. 그러나 단순한 질의 전달은 질의가 전달된 경로 상에 존재하는 POI의 총 개수를 이용하기 때문에, k 개의 POI를 검색하기 위해 필요한 셀보다 많은 수의 셀을 검색한다. 이를 해결하기 위해, 모든 서버는 각 셀에 존재하는 POI의 수를 관리하는 셀 테이블을 메모리상에 유지한다. 이를 이용하여 질의 전달에 따른 홉(hop) 수와 검색이 예상되는 셀의 POI 수를 이용하여 검색될 셀의 수를 감소시킬 수 있다.

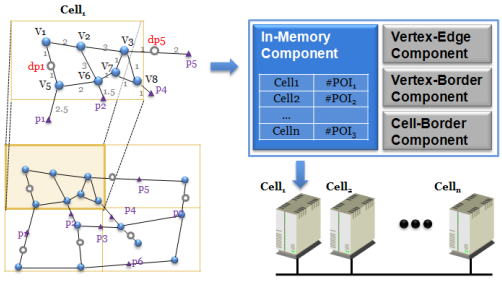


그림 2. 분산 그리드의 전체적인 구조

분산 그리드 기법에서는 공간 네트워크를 그리드 셀로 분할하고 하나의 셀을 하나의 서버가 관리하기 때문에, 네트워크 정보의 변경이 발생하였을 경우 이를 서버에 전달하여 정보를 갱신해야 한다. 첫째, 공간 네트워크에 새로운 노드의 삽입, 삭제, 노드 사이의 거리 변경이 발생할 경우, 변경이 발생한 셀을 담당하는 서버에서 노드 및 경계점 정보를 이용하여 새로운 거리를 계산한다. 이를 반영하여 Vertex-Edge, Vertex-Border, Cell-Border 컴포넌트의 정보를 갱신한 후, 경계점의 위치 변경이 발생할 경우에만 이를 공유하는 셀의 담당 서버로 전송하여 정보를 갱신하도록 한다. 둘째, 이동 객체는 셀별로 R 트리를 이용하여 색인한다. 이때, 해당 셀에 새로운 이동 객체가 진입하거나 기존 이동 객체가 다른 셀로 이동하였을 경우, 셀별 R 트리에서 이동 객체의 삽입 및 삭제를 통해 위치변경을 반영한다. 아울러, 셀내에 존재하는 POI의 수의 변경을 방송(broadcasting) 메시지를 통해 모든 서버로 전달하여 셀 테이블 정보를 갱신한다. 분산 그리드 기법에서는 공간 네트워크 정보를 POI 정보와 독립적으로 관리한다. 따라서 POI의 정보관리는 R 트리와 같은 색인 구조를 이용하여 독립적으로 관리할 수 있다.

3.2 분산 그리드를 위한 k-최근접 질의 처리 알고리즘

분산 그리드 기법에서 k-최근접 질의 처리는 S-GRID와 달리 k 개의 POI를 검색할 때까지 이웃 셀을 관리하는 서버에 질의를 전달하여 POI를 검색하고, 결과를 전달받아 취합하는 과정이 필요하다. 이때, 취합된 결과의 수가 k 보다 적을 경우, 질의를 재전송하여 k 개의 POI를 검색할 때까지 질의 전송과 결과의 수신이 반복되어 검색 성능이 저하된다. 이를 해결하기 위해, 모든 서버는 각 셀에 포함된 POI의 수를 저장하는 In-memory 셀 테이블을 유지한다. 질의점이 발생한 서버에서는 해당 서버에서 POI를 검색한 후, k 번째 POI의 거리보다 작은 경계점을 공유하는 셀의 담당 서버로 질의를 전달한다. 질의를 전달받은 서버에서는 홉(hop) 수를 증가시킨 후, 질의 서버로부터 자신의 셀까지의 질의 전달 홉 수 및 In-memory 셀 테이블을 이용하여 검색이 예상되는 POI의 수를 계산한다. 일반적으로 질의가 바로 인접 셀로 전달된다고 가정할 경우, 질의를 전달받은 셀에서 POI 검색 완료시 전체적으로 검색이 예상되는 POI 수는 식 (1)과 같다.

$$\#ofPOI_n = \sum_{i=-n}^{n} \sum_{j=-|n-|i||}^{|n-|i||} \#ofPOI(i,j) \qquad \text{식 (1)}$$

그림 3은 식(1)을 이용하여 검색이 예상되는 POI의 수를 계산하는 예를 나타낸다. 질의점이 위치한 셀의 좌표가 (2,2)라 가정하고, 질의를 전달받은 서버가 담당하는 셀의 좌표가 (3,3)이라 가정하자.

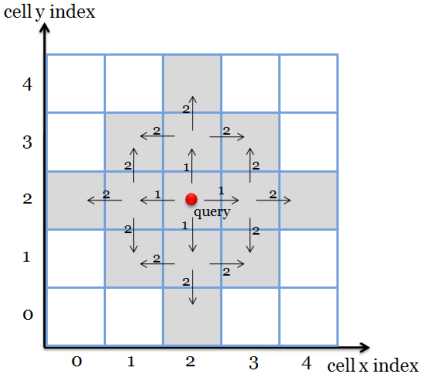


그림 3. 검색될 POI의 수 계산의 예

셀 (3,3)을 담당하는 서버는 질의된 포함된 홉 수가 2이므로 회색으로 표시된 셀들이 질의를 전달받아 질의처리를 수행할 것으로 예상할 수 있다. 따라서 홉 수 2 및 In-Memory 셀 테이블을 이용하여 회색으로 표시된 셀들의 POI 수를 합하여 검색이 예상되는 POI의 수를 계산한다. 이때, 검색된 POI의 수가 k보다 적은 경우, 인접한 모든 셀의 담당 서버로 질의를 재전송하며, 그렇지 않으면 질의를 전송한 서버로 결과를 전달한다. 질의점이 발생한 서버는 질의를 전송한 모든 서버로부터 결과를 받고, 거리 순으로 정렬하여 질의점으로부터 가장 가까운 k 개의 POI를 후보 집합에 삽입한다. 아울러, 전송받은 결과로부터 k 번째 POI까지의 거리보다 작은 경계점을 검색하여, 해당 경계점을 공유하는 셀들로 질의를 재전송한 후 결과를 통합하여 반환한다. <그림 4> 는 k-최근접 질의 처리 알고리즘을 나타낸다.

k-최근접 질의 처리 알고리즘은 크게 내부 확장과 외부 확장의 두 단계로 구성된다. 내부 확장은 질의점이 존재하는 셀의 담당 서버에서 이루어지며, 외부 확장은 그 외의 서버에서 이루어진다. k-최근접 질의 처리 알고리즘은 두 개의 우선순위 큐인 Qv, Qdp 를 이용한다. Qv는 공간 네트워크의 노드 ID 및 경계점과 질의점으로부터의 거리순서로 유지되며, Qdp는 질의점으로부터 검색된 POI ID와 POI까지의 거리순서로 관리한다. k-최근접 질의 처리 알고리즘의 전체적인 수행 과정은 다음과 같다. 먼저, 질의점 q의 좌표를 이용하여 질의가 위치한 셀인 경우, 내부 확장을 수행하며(3-26 라인), 그렇지 않으면 외부 확장을 수행한다(28-44 라인). 내부 확장의 경우, 첫째, 질의점을 포함하는 에지 상의 POI를 검색하여 Qdp에 삽입하고(3 라인), 질의가 속한 에지의 양 노드 및 질의 점으로부터 셀의 모든 경계점까지의 거리를 Qv에 삽입한다(4-6 라인). 둘째, Qv에서 확장을 시작할 노드를 검색한다(8 라인). 검색된 정보가 노드인 경우, vertex-edge 컴포넌트로부터 인접 노드를 검색하고, 인접 노드 및 해당 에지 상에 존재하는 POI를 Qv와 Qdp에 각각 삽입한다(9-13 라인). 검색된 정보가 경계점인 경우, 질의를 전송할 셀 리스트에 경계점을 공유하는 셀 정보와 해당 셀에서 외부 확장을 위한 경계점 정보를 삽입한다(14-16 라인). 이때, 셀 리스트는 질의점이 속한 셀의 좌표, 질의가 전송될 셀의 좌표, 질의가 전송될 셀과 공유하는 경계점 정보로 구성된다. 셋째, 앞의 과정을 셀 내의 모든 POI 또는 모든 노드가 검색되었거나 k 번째 POI 보다 거리가 작은 노드를 모두 검색할 때까지 반복 수행한다. 넷째, 셀 리스트에 존재하는 모든 인접 셀들로 질의를 전송한 후 검색 결과를 전달받는다(20-21 라인). 전달받은 결과로부터 검색된 POI 정보를 Qdp에 삽입한다(22 라인). 이때, k 번째 POI보다 거리가 작은 경계점이 존재할 경우, 해당 경계점의 인접 셀로 질의 재전송을 위해 셀 리스트를 구축하고(24-26 라인), 질의 전송 및 결과 통합을 수행한다. 마지막으로, 검색된 k 개의 POI 를 반환하여 내부 확장을 종료한다. 한편 질의가 위치한 셀이 아닐 경우, 외부 확장을 수행한다(28-44 라인). 외부확장은, 첫째,

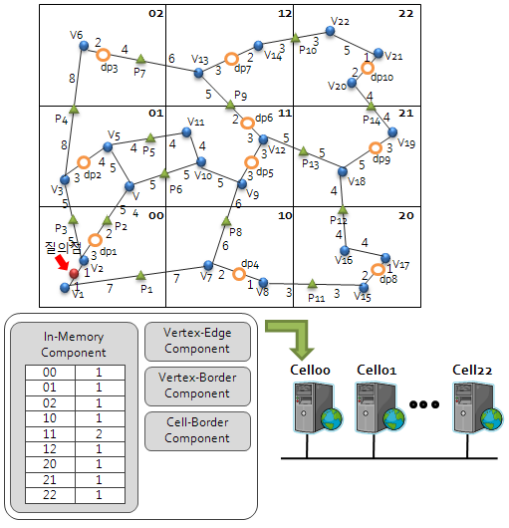
```
ICE(q, k, Query, Result)
Qv=∅, Qdp=∅
 01. qCell=findCell(q)
 02. if (myCell==qCell)
 03. for each POI∈findPOI(q,e)
     Qdp.update(POI, dist(q, POI))
 04. for each v∈{q.e.vs, q.e.ve }
     Qv.update(v, dist(q, v))
 05. for each bp∈myCell.BP
     Qv.update(bp, dist(q, bp))
 06. dMax=Qdp.dist(k)
 07. do
 08. vx=Qv.deque, mark vx as visited
 09. if (vx is a vertex)
 10. for each adjacent vertex vy of vx
 11. for each POI∈findPOI(ex,y)
     Qdp.update(POI,dist(q,POI))
 12. Qv.update(vy, dist(q,vy))
 13. if (all POI in myCell is discovered or
     Qdp.maxdist()<dist(q,vx)) break
 14. else
 15. for each Celli∈findCells(vx)
 16. if (Celli≠myCell)
     CellList.update(myCell,Celli,vx,
     dist(q,vx))
 17. dMax=Qdb.dist(k)
 18. while( d(q, vx) < dMax && Qv≠∅ )
 19. while( CellList≠∅ )
 20. for each Celli∈CellList
     SendQuery to Celli
 21. for each Celli∈CellList
     ReceiveResult from Celli
 22. for each POI in Result.POI from Celli
     Qdp.update(POI, dist(q,POI))
 23. dMax=Qdp.dist(k)
 24. for each vy∈Result.BP from Celli and
     dist(q,vy) < dMax
 25. Cellj=findCell(vy)
 26. if (Cellj≠Celli)
     CellList.update(myCell,Cellj,vy,dist(q,vy))
 27. else
 28. for each bpi∈Query.BP
 29. for each bpj∈myCell.BP
 30. if (bpi≠bpj)
     Qv.update(bpj,dist(q,bpj))
 31. for each POI∈myCell
 32. Qdp.update(POI,dist(q,POI))
 33. dMax=Qdp.dist(k)
 34. bp=Qv.deque
 35. while( dist(q,bp) < dMax )
 36. for each Celli∈findCells(bp)
 37. if (Celli≠myCell)
     CellList.update(myCell, Celli, bp, dist(q,bp))
 38. while( CellList≠∅ )
 39. for each Celli∈CellList
     SendQuery to Celli
 40. for each Celli∈CellList
     ReceiveResult from Celli
 41. for each POI∈Result.POI from Celli
 42. Qdp.update(POI, dist(q,vx)+dist(vx, POI))
 43. Result.update(Qdb, Qv)
 44. SendResult to Query.fromCell
 45. return POIs in Qdp
```

그림 4. k-최근접 질의 처리 알고리즘

전달받은 질의에 저장된 경계점으로부터 셀 내의 모든 경계점까지 거리를 Qv에 저장한다(29-30 라인). 둘째, 셀 내에 존재하는 모든 POI에 대해 POI가 존재하는 에지의 양쪽 노드와 경계점까지의 거리를 Vertex-Border 컴포넌트로부터 검색하고, POI까지의 거리를 계산하여 Qdp에 삽입한다(31-32 라인). 셋째, 식 (1)을 이용하여 해당 셀까지 검색이 예상되는 POI의 수를 계산한다. k 보다 큰 경우 거리가 가장 큰 POI의 거리로 dMax를 정하며 그렇지 않은 경우는 ∞로 정한다(33 라인). 넷째, dMax 보다 작은 거리를 갖는 경계점을 검색하고, 해당 경계점의 인접 셀로 질의를 전송하기 위해 셀 리스트를 생성한다(35-37 라인). 다섯째, 셀 리스트에 존재하는 모든 인접 셀들로 질의를 전송한 후 결과를 통합한다(38-43 라인). 마지막으로, 질의를 전달받은 셀로 결과를 재전송하여 외부 확장을 종료한다.

3.3 k-최근접 질의 처리 알고리즘의 예제

본 절에서는 분산 그리드 기법의 k-최근접 질의 처리 알고리즘의 예제를 제시한다. <그림 5>는 알고리즘의 예를 위한 공간 네트워크와 질의처리 과정을 나타낸다. 공간네트워크는 고정 크기의 3*3 그리드 셀로 나누어진다. 또한 하나의 그리드 셀 정보는 하나의 서버에서 관리하며 자신의 셀 정보인 In-Memory 셀 테이블, Vertex-Edge, Vertex-Border, Cell-Border를 유지한다. N은 공간 네트워크의 노드를 나타내며, P는 셀과 셀의 경계점을 나타낸다. 사용자의 질의가 "현재위치(질의점)에서 가까운 5개의 레스토랑을 찾고자 한다." 라고 가정하자. 먼저, 질의 점의 좌표를 이용하여 질의점이 속하는 셀 Cell 00을 담당하는 서버로 질의를 전달하여, k-최근접 질의 처리 알고리즘을 수행한다. 첫째, 질의 점으로부터 질의가 속한 에지의 양쪽 노드 및 셀의 모든 경계점까지의 거리를 이용하여 Qv를 생성한다. 둘째, Qv에서 확장할 노드인 v1을 불러온다. v1은 노드이기 때문에 내부 확장을 수행하며, Vertex-Edge Component를 이용하여 인접 노드인 경계점 (p1,8)을 검색한다. 경계점 p1은 Qv에 저장된 거리와 동일하므로 변경되지 않는다. 노드 v1과 경계점 p1이 구성하는 edge에 POI 가 없기 때문에, Qdp는 변경되지 않는다. 셋째, Qv로부터 확장할 다음 노드인 v2를 불러온다. v2는 노드이기 때문에 내부 확장을 수행하며 Vertex-Edge Component를 이용하여 인접한 경계점 (p2,6), 경계점 (p3,6)를 검색한다. 경계점 p2, p3는 Qv에 저장된 거리와 동일하므로 변경되지 않는다. 노드 v2와 경계점 p2가 구성하는 edge에 POI (dp1,4)가 존재하므로 이를 Qdp에 삽입한다. 넷째, Qv로부터 확장할 다음 경계점 p2를 가져온다. p2는 경계점이기 때문에 이를 공유하는 셀 01로 질의를 전송하기 위해 CellList에 저장한다. 같은 방법으로 Qv의 모든 노드 및 경계점을 확장하여 Qdp와 CellList를 갱신한다. 다섯째, CellList에 있는 모든 셀에게 질의전송 셀 번호, 질의를 전달받을 셀 번호, 경계점 식별자, 질의 점으로 부터 경계점까지의 거리, 검색된 POI의 수, k번째 POI까지의 거리, k를 전송한다. 즉, 01번 셀의 담당 서버로는 (00, 01, (p2,p3), (6, 6), 1, ∞, 5), 10번 셀의 담당 서버로는 (00, 10, p1, 8, 1, ∞, 5) 를 전송한다. 여섯째, 메시지를 받은 각 셀의 서버는 Vertex- Border Component를 이용하여 전송받은 질의 메시지의 경계점으로부터 셀 내의 POI까지의 거리를 계산한다. 01번 셀의 경우 (dp2, 8), 10번 셀의 경우 (dp4, 9)를 검색한다.



단계	서버	Qv	Qdp	CellList
1	00	(v_1,1), (v_2,1), (p_2,6), (p_3,6), (p_1,8)	-	-
2	00	(v_2,1), (p_2,6), (p_3,6), (p_1,8)	-	-
3	00	(p_2,6), (p_3,6), (p_1,8)	(dp_1,4)	
4	00	(p_3,6), (p_1,8)	(dp_1,4)	(p_2,01)
5	00	-	(dp_1,4)	(p_2,01), (p_3,01), (p_1,10)
6	01,10		(dp_2,8), (dp_4,9)	
7	02,11, 20		(dp_8,26), (dp_3,29) (dp_5,30), (dp_6,36)	
8	01, 10		(dp_2,8), (dp_4,9), (dp_3,29), (dp_8,26), (dp_5,30), (dp_6,36)	
9	00		(dp_1,4), (dp_2,8), (dp_4,9), (dp_8,26), (dp_3,29), (dp_5,30), (dp_6,36)	
10	00		(dp_1,4), (dp_2,8), (dp_4,9), (dp_8,26), (dp_3,29)	

그림 5. k-최근접 질의 처리 알고리즘의 예

일곱째, In-Memory 셀 테이블을 이용하여 첫번째 확장을 통해 검색될 POI의 수를 계산하고, 인접 셀로의 추가 확장 여부를 판단한다. 그 결과 검색이 예상되는 POI의 수는 3이 되어 k보다 적다는 것을 알 수 있다. 따라서 인접 셀로 확장하기 위해, 확장을 위한 질의 메시지를 구성한다. 01번 셀은 (01, 02, p4, 19, 2, ∞, 5)와 (01, 11,

(p5,p6), (14,11), 2, ∞, 5) 질의 메시지를 경계점을 공유하는 인접 셀 02, 11번 셀로 전송하고, 10번 셀은 (10, 11, p8, 21, 2, ∞, 5)와 (10, 20, p11, 21, 2, ∞, 5)의 질의 메시지를 셀 11, 20번 셀로 전송한다. 여덟째, 메시지를 받은 각 셀의 담당 서버는 Vertex-Border Component를 이용하여 셀 내의 POI 까지의 거리를 계산한다. 02번 셀은 (dp3, 29), 11번 셀은 (dp5, 30), (dp6, 36), 20번 셀은 (dp8, 26)의 POI 를 검색한다. 아홉째, In-Memory 셀 테이블 통해서 두 번째 확장까지 검색될 POI 의 총 수는 7이 되어 k 보다 크다는 것을 알 수 있다. 따라서 추가 확장없이 검색된 결과를 이용하여 (검색된 POI의 수, 검색된 POI, 검색된 POI의 거리)를 결과로 전송한다. 질의 결과 취합은 셀 확장의 역순으로 이루어진다. 마지막으로, 질의점이 속하는 셀은 이를 취합하여 질의점으로부터 가까운 k 개의 POI (dp1, dp2, dp4, dp8, dp3)를 반환한다.

4. 성능 평가

본 장에서는 본 논문에서 제안하는 분산 그리드 기법과 기존의 연구인 S-GRID와 성능평가를 수행한다. 분산 그리드 기법의 구현 환경은 HP ML 150 G3 서버, 인텔 Xeon 3.0 Ghz dual CPU, 2 GB 메모리, HP 250 GB SATA 7200 rpm HDD, BCM5703 Gigabyte Ethernet 환경에서 visual studio 2003을 이용하여 구현하였다. 성능 평가에 사용된 공간 네트워크는 220,000 개의 에지와 170,000 개의 노드로 구성된 샌프란시스코 만 데이터를 이용하였으며, POI는 Brinkhoff 알고리즘[5]을 이용하여 에지 대비 POI 밀도 0.01(2200개), 0.02(4400개), 0.05 (11000개), 0.1(22000개)의 데이터 집합을 생성하였으며, 기존 색인 구조인 R 트리를 이용하여 색인 하였다. 사용된 R 트리의 pan-out은 50이며, 트리의 깊이는 S-GRID의 경우 데이터 집합별로 각각 2, 3, 3, 4이며, 분산 그리드의 경우 평균 1.4의 깊이를 갖는 R 트리가 생성되었다. 성능 평가에 사용된 질의 데이터는 공간 네트워크를 구성하는 노드 중에서 랜덤하게 100개를 추출하여 사용하였으며, 100개의 질의 데이터에 대한 응답시간의 평균값으로 성능 비교를 수행하였다. 분산 그리드 기법은 각 셀을 담당하는 서버가 존재하지만, 본 논문에서는 하나의 서버에서 멀티쓰레드 기법을 이용하여 성능평가를 수행하였다. 하나의 쓰레드는 하나의 셀을 담당하며, 실제 분산 그리드 기법과 유사 환경을 제공하기 위해 쓰레드는 네트워크 통신을 통해 질의 및 결과를 전달하도록 하였다. 성능 평가는 첫째, 그리드 셀 수의 변화에 따른 성능 비교, 둘째, k 값의 변화에 따른 성능 비교, 셋째, POI의 밀도에 따른 성능 비교, 넷째, POI 밀도에 따른 POI 없데이트 성능 비교, 다섯째, POI 업데이트에 따른 검색 성능 비교, 마지막으로 네트워크 통신 및 저장 공간 오버헤드를 측정하였다.

첫째, <그림 6> 은 그리드 셀의 수 변화에 따른 k-최근접 질의 처리 알고리즘의 검색 성능을 나타낸다(S-Grid의 k-최근접 질의 처리 알고리즘의 검색 성능을 S-GRID, 제안하는 분산 그리드를 위한 k-최근접 질의 처리 알고리즘의 검색 성능을 분산 그리드로 명명). 이때, k는 20, POI 밀도는 0.01 이다. 두 방법 모두 셀의 수가 증가할수록 성능이 저하됨을 알 수 있다. 이는 셀의 수가 많아질수록 셀 내에 존재하는 POI의 수가 감소하여 k 개의 POI를 검색하기 위해 필요한 셀의 수가 증가하기 때문이다. 셀의 수가 20 이상일 경우, S-Grid의 검색시간이 제안하는 분산 그리드 기법을 위한 k-최근접 질의 처리 알고리즘보다 더욱 증가함을 알 수 있다. 이는, 첫째, S-Grid의 경우 셀의 수가 증가할수록 경계점의 수가 증가하여 Vertex-Border, Cell-Border 컴포넌트의 검색 시간이 증가하는 반면, 제안하는 k-최근접 질의 처리 알고리즘은 멀티 쓰레드를 이용하여 외부 확장이 단계별로 동시에 일어나기 때문에 검색시간이 상대적으로 완만하게 증가한다. 둘째, S-Grid의 경우 한 개의 R-tree를 이용하여 네트워크 상에 존재하는 모든 POI를 색인하지만, 제안하는 k-최근접 질의 처리 알고리즘의 경우 자신의 셀에 존재하는 POI 정보만을 유지하기 때문에 POI를 검색을 위한 R-tree 탐색 비용이 감소된다.

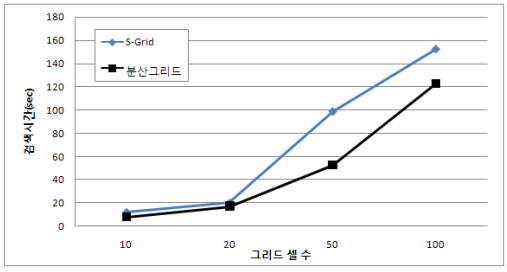


그림 6. 그리드 셀 수에 따른 검색 성능

둘째, <그림 7>은 k 값의 변화에 따른 검색 성능을 나타낸다. 이때, POI의 밀도는 0.01, 그리드 셀의 개수는 20*20이다. S-GRID와 분산 그리드를 위한 k-최근접 질의 처리 알고리즘은 k 값이 증가할수록 성능이 저하됨을 알 수 있다. 이는 k 개의 POI를 검색하기 위한 셀의 수가 증가하여, 보다 많은 외부확장을 수행하기 때문이다. k가 5일 경우, S-Grid의 경우 약 10초, 제안하는 k-최근

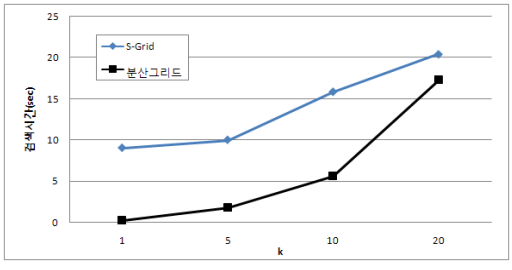


그림 7. k 값의 변화에 따른 검색 성능 비교

접 질의 처리 알고리즘의 경우 약 2초의 검색시간이 소요되어, 제안하는 알고리즘이 보다 우수함을 알 수 있다. 이는 제안하는 알고리즘의 경우 셀 확장에 따른 인접 셀에서의 POI의 검색이 동시에 이루어지기 때문에, POI 검색을 위한 R-tree 탐색 비용이 저하되기 때문이다.

셋째, <그림 8>은 POI 밀도의 변화에 따른 검색 성능을 나타낸다. 이때, k 는 20, 그리드 셀의 개수는 20*20이다. 두 방법 모두 POI 밀도가 증가할수록, 검색 성능이 향상됨을 알 수 있다. 이는 POI 밀도가 증가할수록, k 개의 POI를 검색하기 위한 외부 확장이 감소하기 때문이다. <그림 8>에서 POI의 밀도가 0.01인 경우 S-Grid와 분산 그리드의 k-최근접 질의 처리 알고리즘의 검색 시간은 각각 19초, 17초이다. 아울러 0.1의 경우 제안하는 알고리즘은 약 7초, S-Grid는 약 16초의 검색 시간이 소요되어, POI의 밀도가 증가할수록 제안하는 분산 그리드의 k-최근접 질의처리 알고리즘이 보다 성능이 우수함을 알 수 있다.

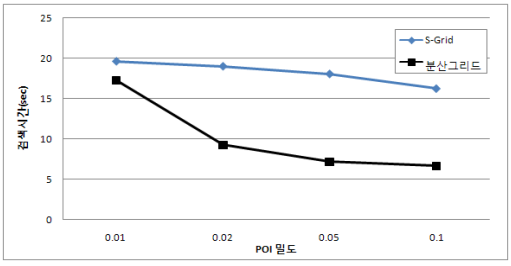


그림 8. POI 밀도 변화에 따른 검색 성능 비교

넷째, <그림 9>는 POI 밀도에 따른 POI의 업데이트 성능 비교를 나타낸다. 이때, 그리드의 셀의 개수는 10*10이고, 전체 데이터 중 10%의 POI 변경이 발생하였을 경우의 업데이트 성능을 측정하였다. 두 방법 모두 POI 밀도가 증가할수록, 업데이트 시간이 증가함을 알 수 있다. 그림에 나타난바와 같이 S-GRID의 경우 전체 데이터에 대해 하나의 R 트리를 이용하여 색인하기 때문에, POI의 밀도가 증가에 따라 업데이트 성능이 급격히 증가하는 반면, 제안하는 분산 그리드 기법은 셀별로 R 트리를 생성하여 색인하여 업데이트가 동시에 처리되기 때문에 업데이트 시간이 완만하게 증가함을 알 수 있다.

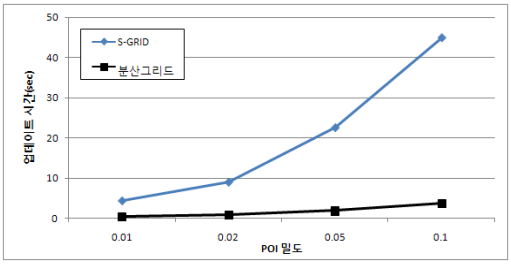


그림 9. POI 밀도 변화에 따른 POI 업데이트 성능 비교

이를 통해 이동 객체와 같이 업데이트가 빈번히 일어날수록, 또한 이동 객체의 수가 증가할수록 업데이트 성능 차이가 더욱 커질 것으로 예상할 수 있다. 따라서 대용량 이동 객체를 효과적으로 지원하기 위해서는 분산 환경으로의 적용이 필수적임을 증명할 수 있다.

다섯째, <그림 10>은 POI 업데이트에 따른 k-최근접 질의처리 성능 비교를 나타낸다. 이때, k는 20, 그리드의 셀의 개수는 10*10, 그리고 전체 데이터 중 10%의 POI 변경이 발생하였을 경우에 검색 성능을 측정하였다. S-GRID의 경우 POI의 업데이트에 따라 POI 밀도가 증가할수록 검색 성능이 저하됨을 알 수 있다. <그림 8>에서 업데이트가 발생하지 않을 경우 POI 밀도가 증가할수록 검색에 필요한 셀의 수가 감소하여 검색성능이 향상되었지만 <그림 10>에 나타난바와 같이 업데이트되는 POI의 수가 증가할수록 업데이트 처리로 인하여 검색 성능이 저하됨을 확인할 수 있다. 제안하는 분산 그리드의 k-최근접 질의처리 알고리즘의 경우 POI의 업데이트를 분산 처리하여 밀도가 0.01에서 0.05까지는 검색 성능이 향상됨을 나타내었다. 이는 업데이트되는 POI의 수가 증가하여도 업데이트의 분산 처리로 인하여 검색 성능에 많은 영향을 미치지 않기 때문이다. 그러나 POI의 밀도가 0.1로 증가하였을 경우 업데이트 성능 저하 폭이 검색 성능 향상 폭보다 커지기 때문에 검색 시간이 완만하게 증가함을 나타내었다. 이를 통해 대용량 이동 객체를 효과적으로 관리하기 위해서는 분산 환경으로의 적용이 필수적이며 효과적인 질의 처리를 위해 S-GRID의 분산 환경으로의 적용이 타당함을 증명할 수 있다.

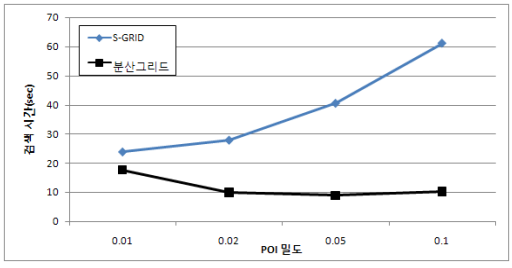


그림 10. POI 업데이트에 따른 검색 성능 비교

마지막으로, 제안하는 분산 그리드의 k-최근접 질의처리 알고리즘의 네트워크 통신 및 저장 공간 오버헤드를 측정 하였다. <그림 11>은 제안하는 분산 그리드의 k-최근접 질의처리 알고리즘의 네트워크 통신 오버헤드를 나타낸다. 이때, k 는 20, POI의 밀도는 0.01이다. 그림에 나타난바와 같이 그리드 셀의 수가 증가할수록 네트워크 통신 오버헤드가 증가함을 알 수 있다. 이는 셀의 수가 증가할수록 셀내에 존재하는 POI 수가 감소하여 k 개의 POI를 검색하기 위해 보다 많은 셀을 탐색함으로써 네트워크 통신이 증가하기 때문이다. 이때, 셀의 수가 20과 50일 경우 검색시간 대비 네트워크 통신 오버헤드는 비슷하지만, 실제 네트워크 통신에 소요되는 시간은 20일

경우 0.8초, 50일 경우 3.5초로 증가함을 확인할 수 있었다. 그러나 전체적으로 검색 시간 대비 5% 미만의 오버헤드가 필요하여 셀의 수가 증가하여도 전체적인 검색 성능에 적은 영향을 미치는 것을 알 수 있다. <표 1>은 S-GRID와 분산 그리드의 저장 공간 오버헤드 비교를 나타낸다. 분산 그리드는 S-GRID와 동일한 구조를 사용하기 때문에 전체 저장 공간 오버헤드가 동일함을 알 수 있다. 그러나 분산 그리드의 경우 서버가 담당하는 셀의 정보만을 저장하기 때문에 평균적 저장 공간 오버헤드는 미미함을 알 수 있다.

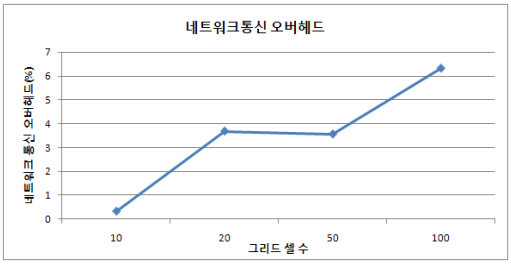


그림 11. 검색 시간 대비 네트워크 통신 오버헤드

표 1. 그리드 셀 수에 따른 저장 공간

(단위 : MByte)

비교항목 / 그리드셀 수	S-GRID	분산 그리드	
		전체 저장 용량	셀별 저장 용량
10*10	1,163	1,163	11.63
20*20	897	897	2.24
50*50	516	516	0.20
100*100	320	320	0.03

5. 결론

본 논문에서는 기존 pre-computation 기법들의 단점을 극복하기 위해, S-GRID를 이용하여 대용량 이동객체의 위치 정보를 효율적으로 관리하기 위한 분산 그리드 기법을 설계하고, 이를 위한 k-최근접 질의처리 알고리즘을 제안하였다. 아울러, 제안하는 분산 그리드 및 S-GRID의 k-최근접 질의처리 알고리즘을 그리드 셀의 수, k, POI의 밀도 변화에 따라 성능 평가를 수행하였다. 성능 평가를 통해, 제안하는 알고리즘이 기존 S-GRID의 알고리즘보다 성능이 우수함을 입증하였다.

향후 연구로는 첫째, k 개의 POI를 검색하기 위해 필요한 셀의 수를 미리 계산하고, 해당 셀들로 질의를 동시에 멀티캐스팅(multicasting)하는 알고리즘을 설계하여 성능을 개선하는 것이며, 둘째, 분산 환경에서 이동 객체를 효율적으로 관리하기 위한 색인 구조와 이를 이용한 연속 k-최근접 질의처리 알고리즘을 개발하는 것이다.

참 고 문 헌

[1] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao, “Query Processing in Spatial Network Databases,” In Proc. of VLDB, pages 802-813, 2003.

[2] M. Kolahdouzan, C. Shahabi, “Voronoi-based Nearest Neighbor Search for Spatial Network Databases,” In Proc. of VLDB, pages 840-851, 2004.

[3] X. Huang, C.S. Jensen, S. Saltenis, “The Islands Approach to Nearest Neighbor Querying in Spatial Networks,” In Proc. of SSTD 2005, LNCS 3633, pp. 73-90, 2005.

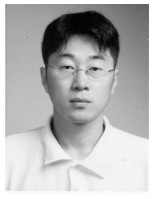
[4] H. Hu, D.L. Lee, J. Xu, “Fast Nearest Neighbor Search on Road Networks,” In Proc. of EDBT 2006, LNCS 4254, pp. 186-203, 2006.

[5] X. Huang, C.S. Jensen, H. Lu, S. Saltenis, “S-GRID: A Versatile Approach to Efficient Query Processing in Spatial Networks,” In Proc. of SSTD 2007, LNCS 4605, pp. 93-111, 2007.

[6] http://www.hyundai-motor.com/Data_Down/pr/cars/ 2007/02_04_01.pdf.

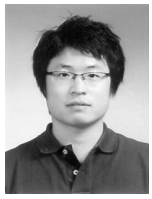
[7] 이호, 이충우, 이준우, 황재일, 박승용, 나연묵, “분산 이동 객체 데이터베이스를 위한 과거 위치 정보 관리,”공간정보시스템학회 논문지, 제8권, 제2호, pp 91-107, 2006.

[8] Y. Nha, T. Wang, K.H. Kim, M.H. Kim, Y.K. Yang, “TMO-structured Cluster-based Real-time Management of Location Data on Massive Volume of Moving Items,” In Proc. of Workshop on Software Technologies for Future Embedded System(WSTFES), pp 89-92, 2003.



김 영 창

2001년 전북대학교 컴퓨터공학과(공학사)
2003년 전북대학교 대학원 컴퓨터공학과(공학석사)
2004년~현재 전북대학교 대학원 컴퓨터공학과(박사과정)
관심분야는 데이터 마이닝, 공간 데이터베이스, 공간색인 구조, 질의처리 알고리즘



김 영 진

2007년 전북대학교 컴퓨터공학과(공학사)
2007년~현재 전북대학교 대학원 컴퓨터공학과(석사과정)
관심분야는 공간 데이터베이스, 센서 네트워크, 색인 구조

장 재 우

1984년 서울대학교 전자계산기공학과 (공학사)
1986년 한국과학기술원 전산학과 (공학석사)
1991년 한국과학기술원 전산학과 (공학박사)
1996년~1997년 Univ. of Minnesota, Visiting Scholar
2003년~2004년 Penn State Univ, Visiting Scholar
1991년~현재 전북대학교 컴퓨터공학과 교수
관심분야는 공간 네트워크 데이터베이스, 위치 기반 서비스, 상황인식, 하부저장구조