이동객체를 위한 분산 그리드 기법의 설계
Design of Distributed Grid Scheme for Moving Objects+

홍승태* · 김영창 · 장재우
Seung-Tae Hong* · Young-Chang Kim · Jae-Woo Chang
전북대학교 전자정보공학부
{sthong*, yckim, jwchang}@dblab.chonbuk.ac.kr

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. 그러나 기존 연구들은 단일 서버 구조로서 대용량 이동객체의 위치정보를 효율적으로 관리할 수 없다는 단점을 갖는다. 본 논문에서는 대용량 이동객체의 위치정보를 분산된 서버에서 효율적으로 관리하기 위하여 분산 그리드 기법을 설계한다. 아울러 제안하는 분산 그리드 기법을 위한 k-최근접 질의처리 알고리즘을 제시한다.

1. 서 론

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 제안된 경로를 따라 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, 질의처리에 필요한 거리 정보를 미리 계산하여 저장하고, 이를 이용하여 검색성능을 향상시킨 pre-computation 기반 기법이다. 그러나 pre-computation 기반 기법들은 위치정보가 빈번하게 갱신되는 이동객체를 효과적으로 처리하지 못하는 단점을 지닌다. 이러한 단점을 극복하기 위하여, 최근 공간 정보를 POI 정보와 독립적으로 관리하기 위한 S-GRID (scalable grid) 가 제안되었다[5]. S-GRID 는 공간을 일정한 크기의 2차원 그리드 셀로 나누고 각 그리드 셀 영역에 포함되는 공간 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 1,500만대 이상이 운행 중인 것으로 집계되었으며[6], 이러한 대용량의 이동 객체의 위치정보를 효율적으로 관리하기 위해서는 분산 처리 연구가 필수적이다[7]. 따라서 본 논문에서는 대용량의 이동 객체 데이터를 분산된 서버에서 효율적으로 관리하기 위한 분산 그리드 기법을 설계한다. 아울러 제안하는 분산 그리드 기법을 위한 k-최근접 질의처리 알고리즘을 제안하며, 마지막으로 분산 그리드 기법에서 특정 서버의 오류 발생시 지속적인 서비스를 지원하기위한 fail-over 방법을 제안

+ 본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신인력양성사업으로 수행된 연구결과임

한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구인 S-GRID를 소개하고, 3장에서는 이동객체를 위한 분산 그리드 기법과 이를 위한 fail-over 방법 및 k-최근접 질의처리 알고리즘을 설계한다. 마지막으로 4장에서는 결론 및 향후 연구 방향을 제시한다.

2. S-GRID

S-GRID는 기존 pre-computation 기법들의 단점인 POI 및 이동객체의 위치정보 갱신으로 인한 검색성능 저하를 해결하기 위하여 제안되었다[5]. S-GRID는 공간을 2차원의 고정 크기의 그리드 셀로 나누어 저장한다. 각 그리드 셀은 그리드 셀과 에지의 교차점을 포함하여 노드 사이의 거리를 미리 계산하여 저장한다. 저장된 정보는 크게 Vertex-Edge, Vertex-Border, Cell-Border 의 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.

S-GRID의 k-최근접 질의 처리 알고리즘은 크게 내부 확장과 외부 확장으로 구성된다. 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 계산한다. 또한, Vertex-Border 컴포넌트 정보를 검색하여 질의점으로부터 해당 셀에 존재하는 모든 POI까지의 거리를 계산하여 결과 집합에 삽입한다. 마지막으로 질의 처리 알고리즘은 질의점으로부터 확장할 노드까지의 거리가 검색된 k 번째 POI까지의 거리보다 크면 종료한다.

3. 이동객체를 위한 분산 그리드 기법의 설계

모바일 기기 및 무선 통신의 발달로 인한 이동 단말기의 보급 확대로 인하여, 수집 가능한 이동객체 데이터가 급증하였다. 이러한 대용량의 이동 객체의 위치정보를 효율적으로 관리하기 위해서는 분산 처리 연구가 필수적이다. 따라서 본 절에서는 대용량의 이동 객체 데이터를 분산된 서버에서 효율적으로 관리하기 위하여 분산 그리드 기법을 설계한다. 아울러 분산 그리드 기법을 위한 k-최근접 질의처리 알고리즘을 제안한다. 마지막으로 제안하는 기법에서 특정 서버의 오류 발생시에도 지속적인 서비스를 지원하기 위한 fail-over 방법을 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. 이때 고정 크기의 그리드 셀 구조를 이용하면, 이동객체가 한 셀에 집중될 때 각 서버간의 로드 불균형이 현상이 발생한다. 이를 해결하기 위해서 서버가 수용 가능한 이동객체의 수를 threshold 값으로 설정하고 threshold를 초과할 경우, 셀을 분할하는 방법이 필요하다. 이때 이동객의 삽입 및 위치변경으로 셀의 크기 변경이 발생할 경우, 셀 정보를 갱신하기 위하여 셀 내에 포함되는 공간 정보를 다시 계산해야하는 오버헤드가 발생한다. 이를 해결하기 위해 전체 공간을 미리 고정 크기의 그리드 셀로 분할하고 threshold를 초과하

여 셀을 분할하여 서버를 나눌 때 이미 저장된 고정 그리드 크기 단위의 셀 단위로 분할한다. 만약 하나의 셀에 threshold 이상의 이동 객체가 존재할 때에만 셀을 분할하여 공간 정보를 재생성하는 오버헤드를 줄인다. 그림 1는 제안하는 분산 그리드 기법의 전체적인 구조를 나타낸다.

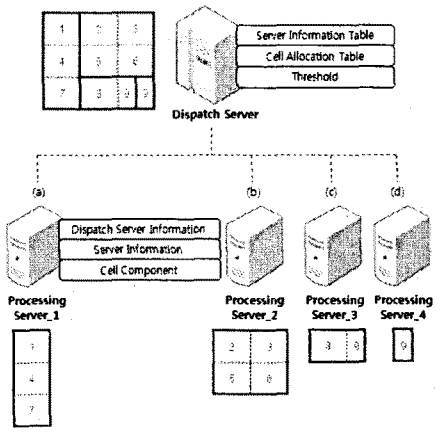

그림 1. 분산 그리드의 전체적인 구조

전체 공간은 고정 크기의 n*n 그리드 셀로 나누어지며, 셀들은 이동객체의 분포에 따라 각 서버에 할당되어 분산 처리하게 된다. 먼저 분산 그리드의 전체적인 구조는 하나의 dispatch 서버와 다수의 processing 서버로 구성된다. 첫째, dispatch 서버는 분산 그리드 기법의 셀 정보 및 각 셀이 할당된 서버의 정보를 server information table과 cell allocation table을 이용하여 관리한다. MO_Threshold는 한 서버가 유지할 수 있는 이동객체의 최대 개수를 나타내며, cell allocation table은 이동객체의 threshold값을 통해 분할된 셀과 해당 셀을 담당하는 서버 정보를 나타낸다. cell allocation table은 {CellID, server information(ServerID, Boundary, #POI)}로 구성된다. CellID는 그리드 셀 ID를 나타내며, server information은 해당 셀이 할당된 서버(ServerID)의 정보를 나타낸다. 마지막으로 #POI는 해당 서버가 유지하고 있는 이동객체의 개수를 나타낸다. 둘째, processing 서버는 분할된 그리드 셀 영역의 공간 데이터의 관리 및 질의를 수행하는 서버이며, {processing Cell List(CellList, Boundary, POIInformation, #POI), Network Data}로 구성된다. processing cell list는 해당 서버에 할당된 셀 리스트와 셀의 범위인 Boundary, 각 셀의 이동객체를 R 트리로 색인하는 POIInformation, 그리고 각 셀의 POI개수를 유지한다. 그리고 Network Data는 셀에 포함된 공간 데이터를 각각 Vertex-Border Component, Vertex-Edge Component, Cell-Border Component 에 유지한다. 분산 그리드의 셀 분할 방법은 다음과 같다. 새로운 이동객체가 삽입되어 하나의 서버가 처리할 수 있는 이동객체의 수인 threshold를 초과하면 셀을 분할한 후, 기존 processing 서버에서 유지하고 있던 분할된 셀들의 정보들을 새로운 서버로 전송하여 처리하도록 한다. 이때 변경된 셀 정보와 processing 서버정보는 dispatch 서버로 전송하여 cell allocation table 정보를 갱신하게 된다. 분산 그리드 기법은 하나의 서버가 처리할 수 있는 이동객체의 수인 threshold를 초과하였을 때, 다음 2가지 경우를 고려하여 동적으로 분할한다. 첫째, 고정된 크기의 그리드 셀 단위로 분할이 가능할 경우, 각 서버에 균등한 수의 이동객체가 할당 되도록 고정 크기의 그리드 셀 단위로 분할한다. 그림 1의 (a), (b)는 고정된 크기의 그리드 셀이 할당된 서버의 예를 나타낸다. 둘째, 하나의 셀에 이동객체가 집중되어 고정 크기의 그리드 셀 단위로 분할할 수 없는 경우, 해당 셀을 분할하여 새로운 두개의 셀을 생성한다. 이때 생성된 셀 정보를 생성하기 위해 공간 정보를 재계산하여 공간 데이터를 재구성한다. 그림 1의 (c), (d)는 하나의 셀에 이동객체가 집중되었을 경우의 셀 분할을 나타낸다. 이러한 분할 방법을 통해 셀이 분할될 때마다 공간 데이터를 재구성하는 overhead를 줄일 수 있다. 또한 동적 분할을 통해 각 서버가 관리하는 이동객체의 수를 균형 있게 유지할 수 있다.

3.2 분산 그리드 기법의 장애 처리

분산 그리드 기법은 공간 정보를 2차원의 그리드 셀 단위로 나누어 저장하고 이를 이동객체의 분포에 따라 여러 서버에 분산되어 처리된다. 이러한 구조는 질의를 효율적이며 빠르게 처리할 수 있는 장점을 가진다. 반면에 분산된 공간 정보를 처리하는 여러 서버중 하나의 서버에 장애가 발생되었을 때 전체 서비스가 중단되는 단점을 가진다. 따라서 본 절에서는 이러한 장애 발생시에도 서비스를 지속할 수 있는 fail-over 방법을 제안한다. 그림 2는 fail-over를 지원하는 분산 그리드의 구조이다. 이동객체가 삽입되어 threshold를 초과하면, 셀을 분할한 후 새로운 서버가 관리할 영역에 대한 정보를 새로운 서버로 전송한다. 이때 현재 서버에 할당되어 있던 셀 영역에 대한 정보를 새로운 서버로 같이 전송하여 두 서버 중 하나의 서버에 장애가 발생하여도 장애 발생 서버의 셀 영역에 대한 질의처리를 수행할 수 있도록 한다. 그림 2의 (a), (b)는 전체 그리드 셀이 유지하고 있는 이동객체의 수가 threshold를 초과하였을 경우의 셀 분할을 나타낸다. 이때 서로의 장애에 대해 대처하기 위해 (a)는 분할되어 (b)로 전송될 2, 3, 5, 6, 8, 9번 셀을 유지한다. 또한 (b)는 (a)로부터 전송받은 받아 담당할 셀들뿐만 아니라 (a)가 초기에 담당하고 있던 1, 4, 7번 셀도 장애처리를 위해 함께 전송받아 유지한다. 마찬가지로 (c)는 (b)로부터 담당할 8, 9번 셀뿐만 아니라 (c)가 초기에 실질적으로 처리하고 있던 2, 3, 5, 6번 셀도 함께 전송받아 유지한다. 이러한 장애 처리 방법을 통해 특정 서버의 장애 시 이웃 서버가 장애가 발생한 서버의 영역을 관리하여 지속적인 서비스를 제공한다.

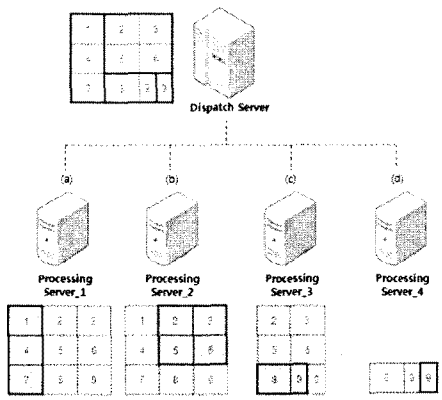

그림 2. 분산 그리드 기법의 장애 처리 방법

3.3 분산 그리드 k-최근접 질의 처리 알고리즘

분산그리드 구조에서 k-최근접 질의처리는 인접셀 또는 인접서버로 확장하면서 수행하게 된다. 분산 그리드 구조는 셀의 이동객체의 분포에 따라 한 서버가 여러 셀들을 관리할 수 있다. 따라서 이웃 셀로 질의를 전송할 때 이웃 셀이 해당 서버에서 처리하는 영역의 경우에는 서버내의 해당 셀을 처리하는 스레드로 질의를 전송하며, 해당 서버에서 처리하는 영역이 아니면 이웃 셀을 담당하는 서버로 질의를 전송한다. 분산그리드 구조에서 k-최근접 질의처리를 위한 순서는 다음과 같다. 첫째, dispatch 서버는 사용자의 질의를 받는다. 둘째, 질의를 통해 dispatch 서버의 Cell Allocation Table을 참조하여 해당 질의점을 포함하는 영역을 처리하는 Processing 서버로 질의를 전송한다. 셋째, 질의를 받은 Processing 서버는 인접 셀 또는 인접 서버로 확장하면서 k-최근접 질의를 처리한다.

분산 그리드 기법에서는 공간 네트워크 정보를 그리드 셀별로 독립적으로 관리한다. 따라서 k 개의 POI를 검색하기 위해서는 질의점이 위치한 셀에서 POI를 검색한 후 이웃 셀에 위치한 POI를 검색하기 위해 다른 스레드 또는 서버로 질의를 전달하여 검색하고 이를 취합하는 과

정이 필요하다. 이때 검색된 POI의 수가 k를 만족하고, k 번째 POI 보다 작은 거리를 갖는 경계점이 없을 때까지 다른 이웃 셀 또는 이웃 서버로 질의를 전달하여 POI의 검색한 후 이를 취합하는 과정을 반복해야 하기 때문에 검색 성능이 저하된다. 이를 해결하기 위해, 모든 서버는 각 셀 안에 존재하는 POI의 수를 유지한다. 질의점이 위치한 셀을 관리하는 서버에서는 해당 셀에 존재하는 POI를 검색한 후, k 번째 POI의 거리보다 작은 경계점을 공유하는 셀의 담당 서버로 질의를 전달한다. 이때 다른 서버는 셀 내의 POI 수를 이용하여 검색이 예상되는 POI의 개수를 계산한다. 이때, 검색된 POI의 수가 k보다 적은 경우, 인접한 모든 셀의 담당 서버로 질의를 재전송하며, 그렇지 않으면 질의를 전송한 서버로 결과를 전달한다. 질의점이 위치한 셀 담당 서버는 질의를 전송한 모든 서버로부터 결과를 받고, 거리 순으로 정렬하여 질의점으로 부터 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 3은 k-최근접 질의 처리 알고리즘을 나타낸다.

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

```
49.        ReceiveResult from Celli
50.      for each (POI in Result.POI from
                                    Celli)
51.        Qdp.update(POI, dist(q,vx)+dist(vx,
                                    POI)) }
52.    Result.update(Qdb, Qv)
53.    SendResult to Query.fromCell
54. return POIs in Qdp
```

그림 3. k-최근접 질의처리 알고리즘

k-최근접 질의 처리 알고리즘은 크게 내부 확장과 외부 확장의 두 단계로 구성된다. 내부 확장은 질의점이 존재하는 셀의 담당 서버의 질의점을 포함하는 셀에서 이루어지며, 외부 확장은 동일 서버에서 질의점을 포함하는 셀 이외의 셀 또는 이웃 서버에서 이루어진다. k-최근접 질의 처리 알고리즘은 두 개의 우선순위 큐인 Qv, Qdp 를 이용한다. Qv 는 공간상의 노드 ID 및 경계점과 질의점으로부터의 거리순서로 유지되며, Qdp 는 질의점으로부터 검색된 POI ID와 POI 까지의 거리순서로 관리한다. k-최근접 질의 처리 알고리즘의 전체적인 수행 과정은 다음과 같다. 먼저, 질의점 q의 좌표를 이용하여 질의가 위치한 셀인 경우, 내부 확장을 수행하며(3-33 라인), 그렇지 않으면 외부 확장을 수행한다(35-53 라인). 내부 확장의 경우, 첫째, 질의점을 포함하는 에지상의 POI를 검색하여 Qdp에 삽입하고(3-4 라인), 질의가 속한 에지의 양 노드 및 질의 점으로부터 셀의 모든 경계점까지의 거리를 Qv에 삽입한다(5-8 라인). 둘째, Qv에서 확장을 시작할 노드를 검색한다(11 라인). 검색된 정보가 노드인 경우, vertex-edge 컴포넌트로부터 인접 노드를 검색하고, 인접 노드 및 해당 에지상에 존재하는 POI를 Qv 와 Qdp 에 각각 삽입한다(9-17 라인). 검색된 정보가 경계점인 경우, 질의를 전송할 셀 리스트에 경계점을 공유하는 셀 정보와 해당 셀에서 외부 확장을 위한 경계점 정보를 삽입한다(18-21 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(24-27 라인). 전달받은 결과로부터 검색된 POI 정보를 Qdp에 삽입한다(28-29 라인). 이때, k 번째 POI보다 거리가 작은 경계점이 존재할 경우, 해당 경계점의 인접 셀로 질의 재전송을 위해 셀 리스트를 구축하고(31-33 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(35-53 라인). 외부 확장은, 첫째, 전달받은 질의에 저장된 경계점으로부터 셀 내의 모든 경계점까지 거리를 Qv에 저장한다(35-37 라인). 둘째, 셀 내에 존재하는 모든 POI에 대해 POI가 존재하는 에지의 양쪽 노드와 경계점까지의 거리를 Vertex-Border 컴포넌트로부터 검색하고, POI까지의 거리를 계산하여 Qdp에 삽입한다(38-39 라인). 셋째, 해당 셀까지 검색이 예상되는 POI의 수를 계산한다. k 보다 큰 경우 거리가 가장 큰 POI의 거리로 dMax를 정하며 그렇지 않은 경우는 ∞로 정한다(40 라인). 넷째, dMax 보다 작은 거리를 갖는 경계점을 검색하고, 해당 경계점의 인접 셀로 질의를 전송하기 위해 셀 리스트를 생성한다(42-44 라인). 다섯째, 셀 리스트에 존재하는 모든 인접 셀들로 질의를 전송한 후 결과를 통합한다(45-51 라인). 마지막으로, 질의를 전달받은 셀로 결과를 재전송하여 외부 확장을 종료한다.

4. 결론 및 향후연구

본 논문에서는 S-GRID를 이용하여, 이동객체의 위치 정보를 효율적으로 관리하기 위한 분산 그리드 기법을 설계하고, 이

를 위한 k-최근접 질의처리 알고리즘을 제안하였다. 아울러, 특정 서버에 장애가 발생하였을 때 이를 처리하기 위한 분산 그리드 기법에서의 장애처리 방법을 제안하였다.

향후 연구로는 설계한 그리드 기법의 구현을 통해 제안한 k-최근접 질의처리 방법의 성능평가를 수행하는 것이다.

참고문헌

[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] 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.