A New k-NN Query Processing Algorithm based on Multicasting-based Cell Expansion in Location-based Services

Seungtae Hong
Dept. of Computer Engineering
Chonbuk National University
Chonju, Chonbuk
danteht@jbnu.ac.kr

Jaewoo Chang
Dept. of Computer Engineering
Chonbuk National University
Chonju, Chonbuk
jwchang@jbnu.ac.kr

Abstract—In telematics and LBS(location-based service) applications, because moving objects usually move on spatial networks, their locations are updated frequently, leading to the degradation of retrieval performance. To manage the frequent updates of moving objects’ locations in an efficient way, we propose a new distributed grid scheme which utilizes node-based pre-computation technique to minimize the update cost of the moving objects’ locations. Because our grid scheme manages spatial network data separately from the POIs(Point of Interests) and moving objects, it can minimize the update cost of the POIs and moving objects. Using our grid scheme, we propose a new k-nearest neighbor (k-NN) query processing algorithm which minimizes the number of accesses to adjacent cells during POIs retrieval in a parallel way. Finally, we show from our performance analysis that our k-NN query processing algorithm is better on retrieval performance than that of the existing S-GRID.

Keywords-component: Distributed grid scheme, query processing algorithm, road network, moving objects

I. INTRODUCTION

With the advancements on GPS and mobile device technologies, it is required to provide location-based services (LBS) to moving objects which move into spatial networks, like road networks [1]. That is, geo-information and geo-processing services are delivered to users with mobile phones, according to their current locations and their points of interests. Such services as automatic vehicle location delivering, tourist services, transport management, and traffic control are all based on mobile objects and the management of their continuous change of location data [2]. Several types of location-dependent queries are significant in LBS, such as range queries [3], k-nearest neighbor (k-NN) queries [3, 4, 5, 6], reverse nearest neighbor queries [7], and continuous queries [8]. Among them, the most basic and important queries are k-NN ones. The existing k-NN query processing algorithms use pre-computation techniques for improving performance [9, 10, 11, 12]. However, when POIs need to be updated, they are inefficient because distances between new POIs and nodes should be re-computed. To solve it, S-GRID [9] divides a spatial network into two-dimensional grid cells and pre-compute distances between nodes which are hardly updated. However, S-GRID cannot handle a large number of moving objects which is common in real application scenario. As the number of moving objects increases, a lot of insertions and updates of location data are required due to continuous changes in the positions of moving objects. Because of this, a single server with limited resources shows low performance for handing a large number of moving objects. To the best of our knowledge, there exists no work to consider a distributed processing technique using multiple servers for spatial networks. Therefore, we, in this paper, propose a distributed grid scheme which manages the location information of a large number of moving objects in spatial networks. Based on our grid scheme, we propose a new k-NN query processing algorithm based on multicasting-based cell expansion which minimizes the number of accesses to adjacent cells during POIs retrieval in a parallel way.

The rest of the paper is organized as follows. In section 2, we present works. In section 3, we describe the details of our distributed grid scheme. Section 4 presents a new k-NN query processing algorithm based on our grid scheme. In section 5, we provide the performance analysis of our k-NN query processing algorithm. Finally, we conclude this paper with future work in section 6.

II. RELATED WORK

In this section, we describe some related works on k-NN query processing in spatial networks. First, Kolahdouzan and Shahabi [9] proposed VN3(Voronoi-based Network Nearest Neighbor) to process k-NN query in spatial networks. VN3 first generates a Network Voronoi Diagram [13] based on a given set of POIs and pre-computes the network distances within each Voronoi polygon. The network expansion within each Voronoi polygon can be replaced by the pre-computed distances. Secondly, Safar [10] proposed a novel approach, termed PINE progressives incremental network expansion), to address KNN queries in SNDB(Spatial Network Databases). PINE partitions a large network into Voronoi cells like VN3 and performs across-the-network computation for only the border points of the neighboring regions. To find the K nearest neighbors, it finds the first nearest POI by simply locating the Voronoi cell that contains a query point q. Then, starting from the query point q, it uses INE algorithm to find POIs. Finally, Huang et al. [11] proposed the Islands approach. Starting from each data point, the Islands approach first pre-computes “islands”. When nodes being within a given radius rmin to the POIs are part of the POI’s island, the distance for such POI is recorded. A k-NN query processing algorithm performs network expansions from the query point by using the pre-computed “islands” encountered during the expansions. Although they reduce the cost of expensive network
expansions by pre-computing network distances between nodes and POIs, they have a disadvantage that the frequent updates of the POI causes performance degradation.

To resolve the problem of the VN3, PINE and Island approaches, Huang et al. [12] proposed S-GRID(Scalable Grid) which represents a spatial network into two-dimensional grids and pre-computes the network distances between nodes and POIs within each grid cell. To process k-NN query, they adopt the INE algorithm [3] which consists of inner expansion and outer expansion. The inner expansion starts a network expansion from the cell where a given query point is located and continues processing until the shortest paths to all data points inside the cell have been discovered or the cell holds no data points. Whenever the inner expansion visits a border point, the outer expansion is performed from that point. The outer expansion finds all POIs in the cells sharing the border point. This process continues until k nearest POIs are found. In S-GRID, the updates of the pre-computation data are local and POI independent. However, S-GRID have a critical problem that it is not efficient in handling a large number of moving objects, which are common in real application scenario, because it focuses on a single server environment. That is, when the number of moving objects is great, a lot of insertions and updates of location data are required due to continuous changes in the positions of moving objects. Thus, a single server with limited resources shows bad performance for handling a large number of moving objects.

III. DISTRIBUTED GRID SCHEME

To support a large number of moving objects, we propose a distributed grid scheme, by extending S-GRID. Our new grid scheme employs a two-dimensional grid structure for a spatial network and performs pre-computations on the network data, such as nodes and edges, inside each grid cell. In our distributed grid scheme, we assign a server to each cell for managing the network data, POIs and moving objects. Each server stores the pre-computed network data and manages cell-level two indices, one for POIs and the other for moving objects. To assign a unique ID(identifier) to each cell, we define CellID as follows.

**Definition 1.** Let assume a spatial network is partitioned into n*n two-dimensional grid structure. A unique ID of a cell being located in i-th row and j-th column, CellID\(_{ij}\), is defined by CellID\(_{ij}\)=\(i\cdot j\).

Figure 1 shows an overall structure of our distributed grid scheme. Each cell consists of seven data structures: a cell table(Cell Table), border point table(BP table), POI R-tree, MO index, Vertex-Edge component, Vertex-Border component, and Cell-Border component.

1. **Cell Table**

   Our distributed grid scheme makes network communication with other servers to access both network data and POI information in a grid cell. For this, each server maintains a cell table whose entry of the cell table is \(<\text{CellIndex}, \text{IP:port}, \#\_\text{POI}>\) where CellIndex is ID of a grid cell, IP:port is a pair of IP address and the port number of the corresponding server, and \#\_\text{POI} is the total number of POIs within the cell boundary.

2. **BP table**

   Our distributed grid scheme maintains BP table to store ID of adjacent cells which share the border point of a cell. The entry of BP table is \(<\text{BPID}, \text{AdjCell}>\) where BPID is ID of a border point in a cell and AdjCell is ID of adjacent cell sharing the border point.

3. **POI R-tree**

   Our distributed grid scheme uses an R-tree to maintain POIs in the cell. While S-GRID has an overhead of a series of split and merge operations due to the update of POIs in a cell, our distributed grid scheme shows good retrieval performance because the update of one cell does not affect the updates of POIs in other cells.

4. **MO index**

   Our MO index consists of edge table and MO table in order to maintain moving objects’ location information. The entry of edge table is \(<\text{EdgeID}, \text{MOID list}>\) where EdgeID is ID of an edge in a cell and MOID list is the list of IDs of moving objects in the related edge. On the other hand, the entry of MO table is \(<\text{MOID, EdgeID, dist, D}>\) where MOID is ID of moving object, EdgeID is ID of an edge containing the moving objects, ‘dist’ is the distance from the starting node of the edge to the moving object, and D is the movement direction of moving object.

5. **Vertex-Edge, Vertex-Border, Cell-Border component**

   Like S-GRID, our distributed grid scheme manages an adjacent node list by using the vertex-edge component, a distance from a node to a border point by using the vertex-border component, and a distance from a border point to another border point by using the cell-border component. Unlike S-GRID, our distributed grid scheme deals with the vertex-edge, vertex-border and cell-border components in a cell-wise manner.

IV. NEW K-NN QUERY PROCESSING ALGORITHM

The k-NN query processing algorithm of S-GRID consists of two steps: inner expansion and outer expansion. First, the inner expansion retrieves POIs by visiting the nodes within a cell in the order of distance from a query point. Secondly, the outer expansion retrieves POIs from adjacent cells sharing border points with the cell in which the query is located. To apply the expansion approach to our distributed grid scheme, the k-NN query processing algorithm should visit adjacent cell repeatedly if there are more than one shared border points between the query cell and the adjacent cell. Let us assume that
server 1 and server 2 manage cell 1 and cell 2, respectively. Because a query point Q is located in cell 1, the algorithm first visits nodes within the cell 1 by using inner expansion. When the next node to visit is border point p2, the algorithm sends the query to server 2 because cell 2 shares the border point. The server 2 computes the distances between the query point and POIs, i.e., p1 and p2, by using the border point p1 provided by the server 1. Then, the server 2 returns the result to the server 1. Next, the server 1 inserts the POIs retrieved from adjacent cells and continues to perform inner expansion. When the next node to visit is a border point p1, the algorithm should send the query to the server 2 again. The server 2 has to repeat the same process as in the case of the border point p2. That is, the process is repeated for the number of shared border points between cell 1 and cell 2. Therefore, the retrieval performance for k-NN query processing is decreased as the number of shared border points increases. To solve this problem, it is necessary to send a query once by maintaining all the shared border points.

For this, we propose multicasting-based cell expansion (MCE) algorithm which sends a query at once to all the servers for managing the cells where the k nearest POIs are located. Our algorithm sends a query to the servers assigned to cells to be visited and finds the k nearest POIs by performing outer expansion in a parallel way. Our MCE algorithm consists of 2 phases. The first phase is to create a list of boundary cells to find the k nearest POIs. The second phase is to send a query to the boundary cells and to find POIs by using outer expansion. First, to create a list of the boundary cells, we compute the number of expected POIs to be retrieved by visiting the adjacent cells. For this, our MCE algorithm finds the number of POIs in each cell by accessing each cell table. Then, it creates a network of border points by using the cell-border component of other cells and expands the network until the sum of POIs within the expanded cells is k. The total number of POIs within the cells can be calculated by using following equation (1). Here, a spatial network consists of n*n grid cells, a query is located in the cell of i-th row and j-th column in the grid, CellId[i,j] is the identifier of a query cell, and ‘hop’ is the number of expansion. Also x and y are the relative row value and the column value between a query cell and a cell visited during the hop-th expansion, respectively.

$$\#POI(CellId[i,j]+n*x+y)$$ is the number of POIs in the cell which is away from a query cell by x rows and y columns.

$$\#POI = \sum_{y=0}^{y=H-1} \sum_{x=0}^{x=H-1} \#POI(CellId[i,j]+n*x+y)$$  \hspace{1cm} (1)

Figure 2 presents a cell list creation algorithm by using the cell-border component of the cells within a boundary. First, the algorithm stores into ‘nCandidate’ the number of POIs in the cell where a query is located, and it inserts the border points of the cell into Qv (line 1-3). Secondly, for a border point from Qv, the algorithm checks whether the cell being shared with the border point is within the boundary or not. If the cell is not within the boundary, the algorithm calls the existing inner and outer expansion algorithms of of S-GRID (line 6-12). Otherwise, it computes the distances from the query to the other border points in the cell and inserts them into Qv (line 13-15). Next, the algorithm stores the cell ID and the distances into the Celllist (line 16) and updates the number of POIs in the cell from the ‘nCandidates’ (line 17-19). Finally, the process is repeated until all the cells within the boundary are expanded or ‘nCandidates’ is k.

**CreateCellList Algorithm**

```
// CreateCellList(q, CellList) // Qv=ϕ, Qdp=ϕ
01. qCell=CellIdfindCell(q) // Query cell
02. nCandidates = CellTable.getnumofPOI(CellId) // Number of POIs
03. for each bp ∈ CellId.BP Qv.update(bp, dist(q, bp)) // Update the Qv list
04. if nCandidates>k dMax=Qv.Dmax() // Update dMax
05. else dMax=∞
06. do
07. bpx=Qv.deque, mark bpx as visited;
08. CellId=FindAdjCell(bp, CellId) // Find adjacent cells
09. if CellId is not within range R
10. CellList.update(qCell, CellId, bp, dist(q,bpx)+dist(bp,bpx)) // Update the CellList
11. CellList.callExpansion(CellId) // Call the expansion algorithm
12. continue
13. for each non-visited bp ∈ CellId.BP
14. if(dist(q,bpx)+dist(bp,bpx)<dMax)
15. Qv.update(bp, dist(q,bpx)+dist(bp,bpx)) // Update the Qv list
16. CellList.update(qCell, CellId, bp, dist(q,bpx)+dist(bp,bpx)) // Update the CellList
17. if(CellId is not visited)
18. nCandidates = CellTable.getnumofPOI(CellId)
19. mark CellId as visited
20. if nCandidates>k dMax=Qv.Dmax() // Update dMax
21. while nCandidates<k && Qv≠ϕ
```

To perform outer expansion, our MCE algorithm creates a list of cells by using the above ‘CreateCellList’ algorithm and sends a query to the relevant cells by multicasting. The cells receiving the query find POIs by using outer expansion. Our MCE algorithm returns the result to the cell where a query is originated. When servers perform outer expansion, they compute the distance between each border point and the nearest node of the edge with POI by using vertex-border component. However, if there are more POIs within a cell than k, the cost of executing outer expansion is higher than that of the network expansion. To solve this problem, we compute the cost of both network expansion and outer expansion and choose one with lower cost to find POIs. The cost of each method can be computed as follows. First, the network expansion expands the spatial network starting from a border point to adjacent nodes by using the vertex-edge component, and it retrieves POIs lying on the expanded edge. Therefore, the cost of network expansion can be computed by

$$\text{COST}_{\text{network-expansion}} = k \cdot E / N$$

Here, N and E are the total number of POIs and edges in a cell, respectively, and k is the number of POIs to be retrieved. Secondly, the cost of outer expansion can be computed by

$$\text{COST}_{\text{outer-expansion}} = B \cdot N \cdot 2^k$$

Here, B is the total number of border points associated with the query and N is the total number of POIs in the cell.

Figure 3 shows the outer expansion algorithm. First, the algorithm computes the cost of network expansion, COSTn, and the cost of outer expansion, COSTo. Secondly, if the value of COSTo is lower than that of COSTn, the algorithm finds POIs by computing the distances between each border point from the BPlist and all of the POIs in the cell (line 4-10). Otherwise, it removes all border points of the BPlist and inserts them into Qv (line 12-13). Thirdly, the algorithm selects a node (or a border point) with the shortest distance from Qv (line 15). Fourthly, if a node is selected, the algorithm inserts adjacent nodes into Qv and stores POIs lying on the adjacent edges into...
Qdp (line 16-20). Otherwise, it inserts border points into BPlist (line 25). Fifthly, the algorithm repeats the above steps until the number of retrieved POIs is k or all the POIs in the cell are found (line 21, 27). Finally, the algorithm returns the retrieved POIs and BPlist to the coordinator handling the cell where the query is given.

Outexpansion Algorithm(q, k, BPlist) // Qdp=φ, Qv=φ
01. COSTne=calculateCOSTne(k, #_Edge, #_POI)
02. COSToe=calculateCOSToe(BPlist, #_BP, #_POI)
03. if(COSToe<=COSTne)
04. for each bpi∈BPlist
05. for each bpj∈Cell-Border Component
06. if (bpi≠bpj)
07. Qv.update(bpj, dist(q,bpi)+dist(bpi+bpj))
08. for each POI∈myCell
09. Qdp.update(POI, dist(q,bpi)+dist(bpi+POI))
10. dMax=Qdp.dist(k), bp=Qv.deque
11. else
12. for each adjacent vertex vx of bp∈BPlist
13. Qv.update(vx, dist(q, bp)+dist(bp, vx))
14. do
15. vx=Qv.deque, mark vx as visited
16. if (vx is a vertex)
17. for each adjacent vertex
18. vy of vx∈Vertex-Edge Component
19. for each POI∈findPOI(ex,y)
20. Qv.update(vy, dist(q,vy)+dist(vx,vy))
21. if (all POI is discovered or Qdp.maxdist()<dist(q,vy))
22. break
23. else
24. BPList.update(vx, dist(q,vx))
25. dMax=Qdb.dist(k)
26. while (d(q, vx) < dMax && Qv≠φ)
27. return POIs in Qdp, bps in Qv

Figure 3. Outer expansion algorithm

Figure 4 shows our MCE algorithm. First, our MCE algorithm creates a list of cells by using CreateCellList algorithm and sends a query to the cells (line 1-3). Secondly, our MCE algorithm finds POIs within the cell holding a query by using inner expansion. Thirdly, our MCE algorithm integrates the partial results from all the cells receiving the query by the coordinator and inserts the integrated result into Qdp (line 5-9). Fourthly, if there is a border point with a shorter distance than k-th POI, our MCE algorithm creates a new list of cells and sends the query to the servers to perform outer expansion (line 10-15). Finally, our MCE algorithm repeats 3-4 steps until we obtain the k nearest POIs and there is no cell in the cell list.

MCE Algorithm(q, k, Query, Result)
// Qv=φ, Qdp=φ, CellList=φ
01. CreateCellList(q, k, CellList)
02. for each celli∈CellList
03. call OuterExpansion(q, k, celli.BPlist)
04. InnerExpansion(q, k, Qv, Qdp, NULL)
05. while( CellList is not empty)
06. for each celli∈CellList
07. ReceiveResult(celli, dplist, celli.BPList)
08. for each POIi∈dplist
09. Qdp.update(POIi)
10. dMax=Qdp.dist(k)
11. for each vy∈celli.BPlist and
   dist(q,vy) < dMax  Cellj=findCell(vy)
12. if(Cellj≠Celli)
13. CellList.update(myCell, Cellj, vy, dist(q,vy))
14. for each celli∈CellList
15. call OuterExpansion(q, k, celli.BPList)
16. return POIs in Qdp

Figure 4. MCE algorithm

V. PERFORMANCE ANALYSIS

We present performance analysis of k-NN query processing algorithm for our grid scheme. We implement our grid scheme by using visual studio 2003 under HP ML 150 G3 server with Intel Xeon 3.0 GHz dual CPU, 2GB memory. In our experiments, we used multiple processes in a single server and each process manages a single cell. To provide an environment appropriate to a distributed grid scheme, we let each process use a different port number to communicate with other processes by using TCP/IP protocol. For spatial network data, we use San Francisco Bay map consisting of 220,000 edges and 170,000 nodes, and generate four sets of POIs (i.e., 2200, 4400, 11000, 22000) by using Brinkhoff algorithm[14]. These POIs are indexed by using R-trees. Moreover, we randomly select 100 nodes from San Francisco Bay map as query points. To measure the retrieval performance of k-NN queries, we average response times for all the 100 query points. Because the existing works VN3[8], PINE[9], island[10] are very inefficient for the update of POIs due to their POI-based pre-computation techniques, they are not appropriate for dealing with a large number of mobile objects in spatial networks. Thus we compare our algorithm with S-GRID algorithm in terms of POI retrieval time.

Figure 5. Retrieval performance
Figure 5(a) first shows the performance of k-NN query processing with the different number of grid cells when k=20 and POI density=0.01. The performance of our algorithm is better than that of S-GRID when the number of grid cells is more than 10^4. This is because our algorithm performs outer expansion in a parallel way. Figure 5(b) shows the retrieval time of k-NN query with the varying value of k when the density of POI is 0.01 and the number of grid cells equals 20*20. It is shown that as the value of k increases, the retrieval times of the two algorithms are increased because when the number of disk I/Os of the cell-border and vertex-border components is increased to visit adjacent cells. We can say form the performance result that our MCE algorithm is better because it can reduce a query propagation step by sending a query to a list of cells at a time.

Figure 6(a) shows the retrieval time of k-NN query with the varying density of POIs where the number of grid cells equals 20*20 and k=20. As a result, we can reduce the cost of inner expansion within a cell and the number of adjacent cells to be visited. Figure 6(b) shows the retrieval time of k-NN query after updating POIs. For this experiment, we measure the search time of k-NN query when the 10% of POIs is updated. In the case of S-GRID, the retrieval time is exponentially increased as the density of POIs increases. This is because S-GRID uses one R-tree to index all the POIs of the network and so the update of POIs in a cell affects the whole system. Whereas, because our grid scheme uses a separate R-tree per each grid cell to index POIs within it, the update of POIs in a cell does not affect all the grid cells globally. As a result, even though the number of updated POIs increases, the retrieval performance of our grid scheme is not dramatically increased.

VI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a grid scheme to manage the location information of a large number of moving objects in spatial networks. Our grid scheme makes use of a node-based pre-computation technique so that it can minimize the update cost of the moving objects' locations. Our grid scheme splits a spatial network into two-dimensional grid cells so that it can update network data locally. Based on our grid scheme, we proposed a new k-NN query processing algorithm. Our algorithm improves the retrieval performance of K-NN queries because it decreases the number of adjacent cells visited by transmitting a query to all the shared border points. Our experimental results show that our algorithm is better on retrieval performance than that of S-GRID. As a future work, we need to extend our grid scheme to handle a spatial network with dense and sparse regions in an efficient manner by using non-uniform grid cells.

ACKNOWLEDGMENT

This research was supported by Korea Institute of Science and Technology Information (KISTI).

REFERENCES

BIOGRAPHIES

Seungtae Hong is in a Ph.D course in the Chonbuk National University. He received the B.S and M.S degrees in Chonbuk National University in 2009 and 2011, respectively. His research interests include security and privacy of database, query processing algorithm, and cloud computing.

Jaewoo Chang is a professor in the Department of Computer Engineering, Chonbuk National University, Korea from 1991. He received the B.S. degrees in Computer Engineering from Seoul National University in 1984. He received the M. S. and Ph. D degrees in Computer Engineering from Korea Advanced Institute of Science and Technology (KAIST) in 1986 and 1991, respectively. During 1996–1997, he stayed in University of Minnesota for visiting scholar. And during 2003–2004, he worked for Penn State University (PSU) as a visiting professor. His research interests include spatial network database, context awareness and storage system.