Distributed k-NN Query Processing for Location Services

Jonghyeong Han[1], Joonwoo Lee[1], Seungyong Park[1], Jaeil Hwang[1], and Yunmook Nah[1]

[1] Department of Electronics and Computer Engineering, Dankook University, Hannam-dong, Yongsan-gu, Seoul, 140-714, Korea
{jhhan, jwlee, sypark}@dblab.dankook.ac.kr, hwangjaeil@yahoo.co.kr, ymnah@dku.edu

Abstract. The architecture named the GALIS is a cluster-based distributed computing system architecture which has been devised to efficiently handle a large volume of LBS application data. In this paper, we propose a distributed k-NN query processing scheme for moving objects on multiple computing nodes, each of which keeps records relevant to a different geographical zone. We also propose a hybrid k-NN scheme, which utilizes range queries instead of k-NN queries for the neighboring overlapped nodes, thus resulting in 30% reduction of query processing cost. Through some experiments, we show the efficiency of hybrid k-NN scheme over naïve k-NN scheme.

Keywords: k-NN query processing, distributed databases, GALIS, location-based services

1 Introduction

Recent advances in location navigation technology and wide distribution of mobile devices have caused rapid growth of interests in Location Based Services (LBS). But, most of the current research activities related with LBS systems are single node-oriented, making it difficult to handle the extreme situation that must cope with a very large volume, at least millions, of moving objects. The architecture named the GALIS (Gracefully Aging Location Information System) is a cluster-based distributed computing system architecture which consists of multiple computing nodes, each dedicated to keeping records relevant to a different geographical zone and a different time-zone [1,2,3]. The GALIS consists of SLDS (Short-term Location Data Subsystem) controlling current location information of moving objects, and LLDS (Long-term Location Data Subsystem) controlling past location information.

To realize location services, we have to support item-based queries, range queries, and k-NN (k-Nearest Neighbor) queries. For a k-NN query, the user specifies a point and the system has to return k closest moving objects. There has been lots of research efforts to efficiently handle k-NN queries, especially for the centralized computing environments [6,7,8,9,10,11,12]. In this paper, we propose a naïve distributed k-NN query processing scheme for moving objects geographically spread over multiple computing nodes. The proposed scheme runs k-NN query for the current node,

modifies the query points for overlapped neighboring nodes, and then executes k-NN queries for neighboring nodes. We have implemented a single node k-NN query processing scheme, by utilizing R-trees [13,14]. We also propose a hybrid k-NN scheme, which utilizes range queries instead of k-NN queries for the neighboring overlapped nodes, thus resulting in 30% reduction of query processing cost.

We propose a naïve method for distributed k-NN query processing on multiple computing nodes in section 2 and a hybrid k-NN query processing scheme in section 3. Some experimental results related with performance and precision of each query processing scheme are shown in section 4. Finally, section 5 concludes the paper.

2 A Naïve Scheme for Distributed Processing of k-NN Queries

The two-dimensional space of interest is divided into n spatial regions and the one-dimensional time axis is divided into p time zones. A region (or partition) of the geographical area dealt with by the LBS system is called a *macro-cell*. Each macro-cell covers a square-shaped region, of which the default unit length is 25.6km but can be set differently. The regions covered by different macro-cells may be of different sizes. With respect to keeping records on current (most recently observed) locations, moving items in a spatial region are covered by a SDP node. With respect to keeping location histories, moving items in a spatial region are handled by up to p LDP nodes. Here, p means the number of time zones (or temporal regions).

2.1 Overall Scheme

A k-NN query can be performed on a single node, as shown in Figure 1(a). For multiple nodes, as shown in Figure 1(b), moving objects for neighboring overlapped nodes have to be considered.

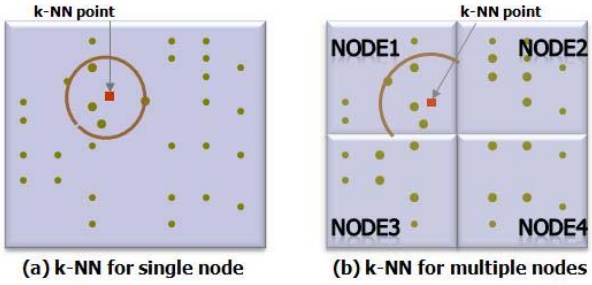

Fig. 1. Comparison of k-NN query on single node and on multiple nodes

We first run the given k-NN query on the current node. If the circle, whose center is the query point (indicated as k-NN point in Figure 1) and whose radius is the distance between the query point and k-th closest point in the current node, is overlapped with neighboring nodes, as shown in Figure 1(b), we have to decide the necessity of broadcasting the queries to the neighboring nodes.

The query processing system to handle k-NN queries consists of Index Creator, Query Analyzer, Query Checker and Query Creator. The structure of the query processing system for the case of 4 computing nodes is illustrated in Figure 2.

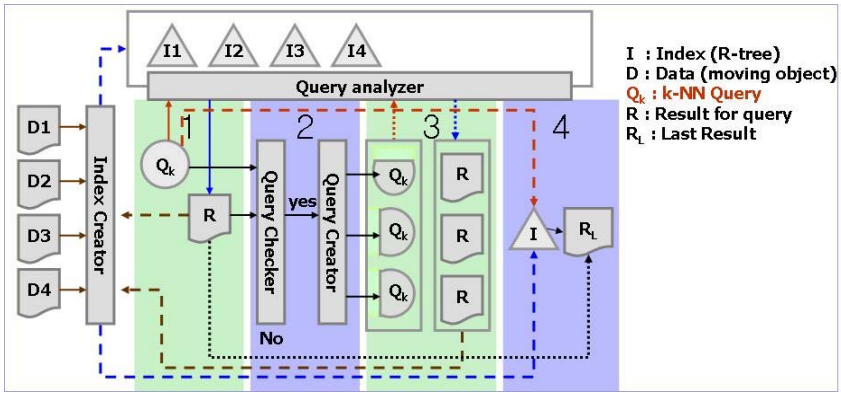

Fig. 2. Query processing system structure for multiple nodes

The Index Creator configures index structures by making use of location information of moving objects stored in each node. The Query Analyzer processes the given queries using appropriate index structures. The Query Checker checks the neighboring nodes based on the query result from the current node, to determine the necessity of transferring the queries to the neighboring nodes. The Query Creator creates a partial query for the target neighboring nodes and sends that partial query to the neighboring nodes. All the query processing results from neighboring nodes are combined and a temporary R-tree index structure is created by the Index Creator. The k-NN query is finally processed by using the final R-tree and its result is returned to user.

2.2 Query Distribution over Neighboring Nodes

If the number of computing nodes is small, it can be possible to process the same k-NN queries over all the nodes and combine all the results of the queries, to process a given k-NN query on a computing node (currently queried node). If, however, the number of the nodes is large, concurrent processing of the same queries over all of the nodes including the current node might result in severe waste of the computing resources.

To reduce computing overhead, we have to determine the neighboring overlapped nodes and transfer the queries to such nodes, if the query point of k-NN query is in the vicinity of the boundary of current node. Figure 3 illustrates such an example of k-NN query on Node 1, where k = 4. In this figure, the object P_2 of Node 2 and the object P_1 of Node 3 can be nearer than the k-NN candidate points between P_{k-3} and P_k on the current node. This means that it is required to send queries to neighboring overlapped nodes and compare candidate locations and related locations to finalize true k-NN points, considering all the nodes. It is expected that enlarging an arc

section of the resultant area for the given k-NN query to a round section causes overlapping of certain part of the area over neighboring nodes.

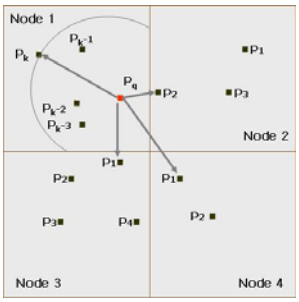

Fig. 3. Necessity of transferring queries to neighboring overlapped nodes

After running the k-NN query on the current node, the neighboring overlapped nodes, which can contain closer points than the current node, can be determined by using the query point and the query results on the current node.

Figure 4 illustrates the procedure to determine the necessity of transferring queries to the neighboring overlapped nodes. Let R be the distance between the query point P_q and the result position P_k (the location of k-th closest object), as shown in Figure 4(a). Let N, S, E, W be the distances between the query point P_q and the boundary points in each direction, P_{bn} (north boundary point), P_{be} (east boundary point), P_{bs} (south boundary point) and P_{bw} (west boundary point), respectively, also as shown in Figure 4(a). The neighboring nodes located in the directions with distances (N, S, E, W) shorter than R, are target nodes to send queries.

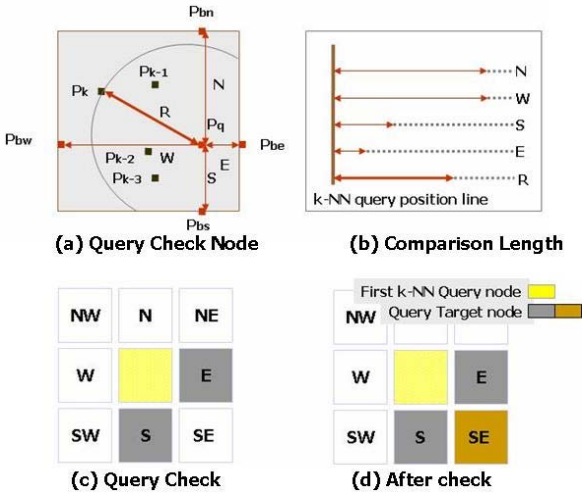

Fig. 4. Determination of neighboring nodes to transfer queries

In Figure 4(b), therefore, the neighboring nodes in south and east direction to the current node are the appropriate nodes to further process the given k-NN query. We

also have to transfer queries to the neighboring node located in the diagonal position to the current node and also adjacent with the selected target nodes. In Figure 4(d), the node in the southeast direction is such a node. Therefore, among 8 neighboring nodes, the final neighboring overlapped nodes, which will further process the given k-NN query, are the nodes in south, east and southeast direction. Algorithm 1 shows the procedure of Query Checker to determine neighboring nodes to send queries.

Algorithm 1. Determining Neighboring Nodes to Send Queries ( )
// P_{bn}, P_{be}, P_{bs}, P_{bw} : boundary points for the current node
// R : distance between the query point and P_k
// N, S, E, W: the distance between the query point and the boundary points
in each direction
// T: target node list
begin
if (N<R) **then**
add north node to T;
if (E<R) add east node and northeast node to T;
if (W<R) add west node and northwest node to T;
else if (S<R) **then**
add south node to T;
if (E<R) add east node and southeast node to T;
if (W<R) add west node and southwest node to T;
else if (E<R) then add east node to T;
else if (W<R) then add west node to T;
else add empty node to T;
endif
end.

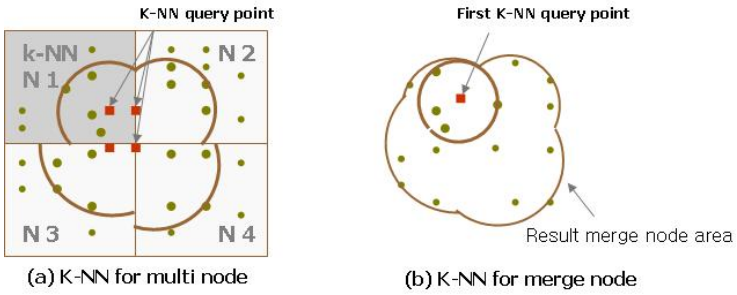

Fig. 5. The modified query points for the neighboring nodes

For the neighboring overlapped nodes, the boundary point (one of P_{bn}, P_{be}, P_{bs} and P_{bw}), which is located on the boundary between the current node and the neighbor node, becomes the new query point. For the neighboring node in the diagonal position, one of the corner point of the current node, facing the neighbor node, becomes the new query point. In Figure 5(a), P_{be} becomes the query point for N2, P_{bs} becomes the query point for N3, and the southeast corner point of N1 becomes the query point for N4. Figure 5(b) shows the typical query shape for the proposed distributed k-NN query processing.

3 Hybrid k-NN Query Processing

By utilizing range queries instead of k-NN queries for the neighboring overlapped nodes, we can reduce the entire query processing time, while obtaining the exactly same query result. We call this slightly modified scheme as the hybrid k-NN query processing scheme. In this hybrid scheme, the Query Checker of the query processing system converts the k-NN query into range queries.

We again use the value R, which is the distance between the query point P_q and the result position P_k (the location of k-th closest object). Let R_n (radius north), R_w (radius west), R_s (radius south) and R_e (radius east) be the same distance with R to the corresponding directions. Let P_{qn} (north query point), P_{qw} (west query point), P_{qs} (south query point) and P_{qe} (east query point) be the virtual points each located in the corresponding direction with distance R. Algorithm 2 shows the procedure of Query Checker to create range queries.

Algorithm 2. Range Query Creation (d direction_of_neighbor_node)
// P_{bn}, P_{be}, P_{bs}, P_{bw} : boundary points for the current node
// P_{qn}, P_{qw}, P_{qs}, P_{qe} : shifted query points to the corresponding direction
// with distance R
begin
if (d=north) **then** create query with range (P_{qw}(x), P_{qn}(y), P_{qe} (x), P_{bn} (y));
if (d=south) **then** create query with range (P_{qw}(x), P_{qs}(y), P_{qe} (x), P_{bs} (y));
if (d=east) **then** create query with range (P_{qe}(x), P_{qn}(y), P_{be}(x), P_{qs}(y));
if (d=west) **then** create query with range (P_{qw}(x), P_{qn}(y), P_{bw}(x), P_{qs}(y));
end.

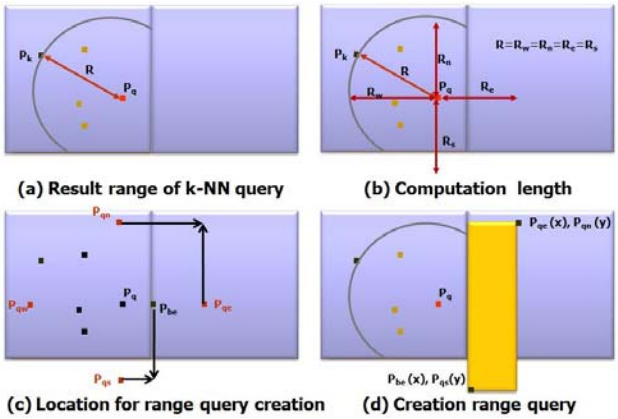

Fig. 6. Range query creation example in the east direction

Figure 6 illustrates the procedure to create the range query for the neighbor node located in the east direction compared to the current node. In this case, the coordinate (P_{qe}(x), P_{qn}(y)) becomes the upper right corner and the coordinate (P_{be}(x), P_{qs}(y)) becomes the lower left corner for the query range.

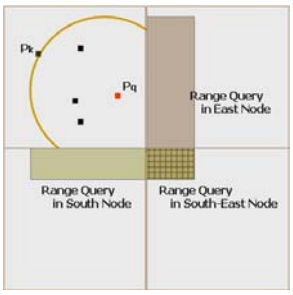

Fig. 7. Query range shape example in hybrid k-NN scheme

The lower part of the query range in the east direction crosses over the node boundary of Node 2. Also, the right part of the range query in the south direction crosses over the node boundary of Node 3. These two out of node query ranges meet at the node in the diagonal direction, Node 4. Therefore, the query range for the diagonal node can be determined from the overflowed portion of one of the neighboring nodes. In our experiment, we use the right or left side neighboring nodes in such cases. Figure 7 shows the query range for the hybrid k-NN scheme.

4 Experiments

Four nodes are configured on a single system for excluding communication delay among the nodes. We used a PC, equipped with 3.0 GHz D-processor and 1 Gbyte memory, with Red Hat FEDARA Core 4 operating system. We generated moving objects by using the object location information generator developed by Marios Hadjieleftheriou of University of California-Riverside. We repeated experiments 18 times, while increasing number of moving objects in each node incrementally, starting from 1,000 objects until reaching 50,000 objects.

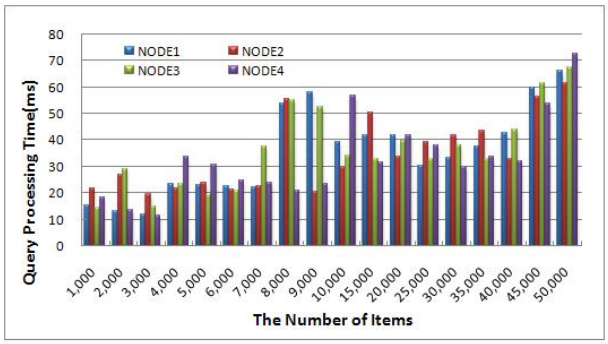

Fig. 8. Naïve k-NN query processing time

To compare query processing times, we measured the processing time of each query 5 times and used the average of 3 measured values, excluding highest and

lowest values. The value k was fixed as 10. Figure 8 shows the query processing time of the naïve k-NN scheme and Figure 9 shows the query processing time of the hybrid k-NN scheme.

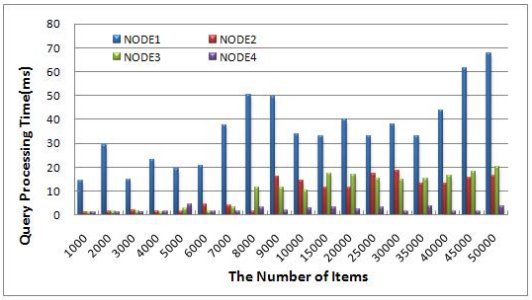

Fig. 9. Hybrid k-NN query processing time

Figure 10 compares the query processing time of naïve k-NN query processing scheme and hybrid k-NN query processing scheme. The query processing times of Node 1 of both methods are the same, because both use the normal k-NN query processing algorithm. But, in other nodes, the query processing time of hybrid scheme is faster than the query processing time of naïve scheme, because the hybrid scheme utilizes range queries instead of k-NN queries for the neighboring overlapped nodes.

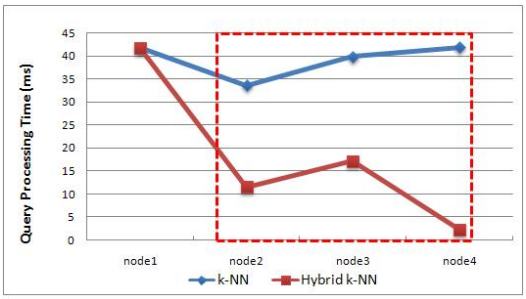

Fig. 10. Comparison of query processing time between naïve and hybrid scheme

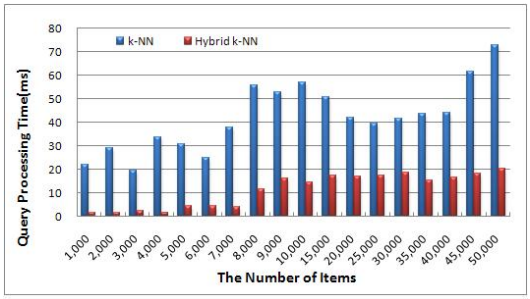

Fig. 11. Comparison of maximum query processing time

Figure 11 compares the maximum query processing times (worst cases) of both schemes. Figure 12 compares the minimum processing time (best case) of naïve k-NN query processing scheme with the maximum processing time (worst case) of hybrid k-NN query processing scheme. It clearly shows that the hybrid scheme is always better than the naïve scheme.

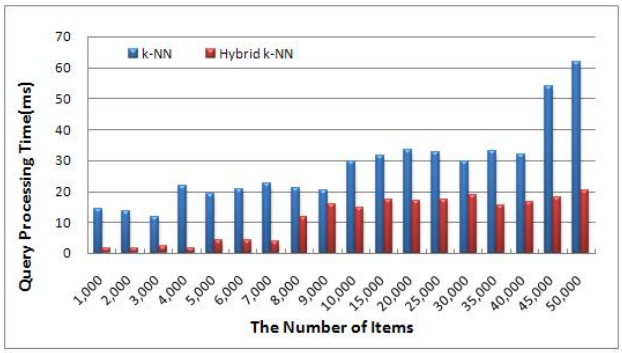

Fig. 12. Comparison of the minimum processing time of naive k-NN query processing scheme with the maximum processing time of hybrid k-NN query processing scheme.

To show the correctness of query results, we compared the query results of k-NN query processing on a single node with 80,000 objects and naïve scheme and hybrid scheme on multiple computing nodes, each having 20,000 objects. The final results of three methods were identical, which means that no correct object is omitted by the naïve and hybrid k-NN query processing schemes.

5 Conclusion

In this paper, we proposed distributed k-NN (k-Nearest Neighbor) query processing schemes for moving objects on multiple computing nodes, each of which keeps records relevant to a different geographical zone. In our naïve method, we first process the k-NN query on the target node, then decide whether the query region is overlapped with other nodes, and finally process the k-NN queries from the shifted query points for the neighboring overlapped nodes.

We also proposed a hybrid k-NN, which utilizes range queries instead of k-NN queries for the neighboring overlapped nodes, thus resulting in low query processing cost, while providing the exactly same query results. Through some experiments, we show the efficiency of hybrid k-NN over naïve k-NN. There should be further experiments on much heavier traffic with millions of moving objects.

Acknowledgments. This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC(Information Technology Research Center) support program, with grant number IITA-2006-C1090-0603-0006 and IITA-2006-C1090-0603-0031, supervised by the IITA(Institute of Information Technology Assessment).

References

1. Nah, Y., Kim, K.H., Wang, T., Kim, M.H., Lee, J., Yang, Y.K.: GALIS: A Cluster-based Scalable Architecture for Location-based Service Systems. Database Research, 18(4). KISS SIGDB (2002) 66-80
2. Nah, Y., Kim, K.H., Wang, T., Kim, M.H., Lee, J., Yang, Y.K.: A Cluster-based TMO-structured Scalable Approach for Location Information Systems. Proc. WORDS 2003 Fall. IEEE CS Press. (2003) 225-233
3. Kim, M.H., Kim, K.H., Nah, Y., Lee, J., Wang, T., Lee, J., Yang, Y.K.: Distributed Adaptive Architecture for Managing Large Volumes of Moving Items. IDPT Vol.2. Society for Design and Process Science (2003) 737-744
4. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. Proc. ACM SIGMOD. (2000) 331-342
5. Zhang, J., Zhu, M., Papadias, D., Tao, Y., Lee, D.: Location-Based Spatial Queries. Proc. ACM SIGMOD. (2003) 467-478
6. Hjaltason, G.R., Samet, H.: Ranking in Spatial Databases. Proc. SSD. (1995) 83-95
7. Cheung, K.L., Fu, A.W.-C.: Enhanced Nearest Neighbor Search on the R-tree. SIGMOD Record 27(3). ACM (1998) 16-21
8. Iwerks, G.S., Samet, H., Smith, K.P.: Continuous k-Nearest Neighbor Queries for Continuously Moving Points with Updates. Proc. VLDB. (2003) 512-523
9. Dasarathy, B.V.: Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. (1991)
10. Song, Z., Roussopoulos, N.: K-Nearest Neighbor Search for Moving Query Point. Proc. SSTD. (2001) 79-96
11. Hadjieleftheriou, M., Hoel, E.G., Tsotras, V.J.: SaIL: A Spatial Index Library for Efficient Application Integration. GeoInformatica 9(4). (2005) 367-389
12. Shakhnarovish, Darrell, Indyk (ed.): Nearest-Neighbor Methods in Learning and Vision. The MIT Press (2005)
13. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. Proc. ACM SIGMOD. (1984) 47-57
14. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-Tree : an Efficient and Robust Access Method for Points and Rectangles. Proc. ACM SIGMOD. (1990) 322-331