MobiPLACE*: A Distributed Framework for Spatio-Temporal Data Streams Processing Utilizing Mobile Clients' Processing Power

Victor Zakhary, Hicham G. Elmongui(✉), and Magdy H. Nagi

Computer and Systems Engineering, Alexandria University, Alexandria, Egypt
{victorzakhary,elmongui}@alexu.edu.eg, magdy.nagi@ieee.org

Abstract. The problem of continuous spatio-temporal queries' processing was addressed by many papers. Some papers introduced solutions using single server architecture while others using distributed server one. In this paper, we introduce MobiPLACE*, an extension to PLACE* [13] system, a distributed framework for spatio-temporal data streams processing exploiting mobile clients' processing power. We will extend the Query-Track-Participate (QTP) query processing model, introduced as a system architecture in PLACE*, by moving the Query server role to mobile clients. This will reduce memory and processing load on our regional servers in exchange for a little additional communication and memory load on mobile devices. This makes the system more scalable and enhances average query response time. Improvements in mobile devices' and communication links' capabilities encouraged us to introduce this extension. In this paper, we will focus on range and k-NN continuous queries and their evaluation on MobiPLACE*. Experimental study is made to compare between MobiPLACE* and PLACE* in terms of server response time and memory.

1 Introduction

Location detecting devices is now wide spread in wide range devices e.g. (mobile phones, cars and many moving devices). Those objects can send their location updates periodically to servers and these data can be used to solve navigation and many location aware services' problems. Such problems can be solved by a system doing continuous queries over those data streams. One of the main challenges to those systems is **scalability**. Mainly, it is about how to design the system and distribute load between system components to scale up to support larger number of moving objects and continuous queries.

Previously, system designers depended on servers to handle all query processing because mobile clients' were poor in capabilities. Nowadays, smart phones, embedded devices in cars and all mobile devices have a PC like capabilities. Designing a system to utilize those capabilities will lead to reduce server loads and allow the servers to handle more client objects, queries and respond with queries' answers shortly. We will focus on continuous range and k-NN (k Nearest Neighbor) queries on objects moving over road networks. In a range query,

I. Stojmenovic et al. (Eds.): MOBIQUITOUS 2013, LNICST 131, pp. 78–88, 2014.
DOI: 10.1007/978-3-319-11569-6_7

the issuer requires to find objects e.g., taxis or clients of taxis in a certain range. In a k-NN query, the issuer requires to find the nearest k objects to her location. We focused on continuous queries instead of snapshot ones. Also, we considered query's incremental evaluation instead of periodic re-evaluation solution. In incremental evaluation, the result is calculated once and saved on either client, server or both and updates to it is only calculated and sent. In periodic re-evaluation, the result is re-calculated from scratch periodically. Incremental evaluation is advantageous because it leads to less tracking and response time but requires some additional memory to save query's result. We need a real-time responses for the query so we chose to save moving objects' locations and do all our computations in memory instead of hard drive. We also assumed that our objects are restricted to move on road networks. This means that the shortest path between object will be used instead of Euclidean distance as a distance metric.

The development of devices' capabilities and mobile broadband services, either in bandwidth or in cost, over the world encouraged us to migrate a server role to mobile clients. This will reduce load on servers with a little increase on communication messages. The ICT 2013's report for mobile broadband service development mentioned that by early 2013, the price of an entry-level mobile-broadband plan represents between 1.2 and 2.2 % of monthly GNI p.c. in developed countries and between 11.3 and 24.7 % in developing countries, depending on the type of service.

In this paper, we extend the work done in [11]. In [11], Sallam applied a modified version of the Incremental Monitoring Algorithm (IMA) [9] on PLACE* [13] Query-Track-Participant (QTP) query processing model to make a **distributed** processing of continuous spatio-temporal queries over **road networks**. We applied the same algorithms by Sallam but on MobiPLACE* QTP query processing model. The difference between the 2 models is in the role distribution of query processing steps between clients' devices and servers.

The rest of this paper is organized as follows. In Sect. 2, we highlighted the related work and explained the systems that we extend in this paper. We gave an overview about MobiPLACE* architecture and communication messages in Sect. 3. We explained how could we execute continuous range and k-NN queries in our system in Sects. 3.3 and 3.4 respectively. Performance evaluation and experiments made were introduced in Sect. 4. Finally, the paper is concluded in Sect. 5.

2 Related Work

Many papers have addressed the problem of continuous queries over spatio-temporal data streams. SINA [7], sets an algorithm to evaluate concurrent continuous spatio-temporal queries. It uses three phases, the hashing phase, the invalidation phase, and the joining phase; to calculate positive and negative updates. SOLE [6], keeps track of only the significant objects in order to save the scarce memory resource. MQM [2] divided the region of study into domains

and object reports its location to server whenever its movement affects any range query results (i.e., crossing any query boundaries) or it changes its current domain. MobiEyes [3] ships some part of the query processing down to the moving objects, and the server mainly acts as a mediator between moving objects. CPM [8], YPKCNN [15], and SEA-CNN [14] introduce 3 algorithms for exact k-NN continuous monitoring in Euclidean space. The above systems assumed a central server architecture for query processing and the Euclidean distance as a distance metric. Some papers considered some restriction on objects motion i.e. (objects are moving in a road network). In this case, the shortest path between objects would be considered as the distance metric. In [4], they designed a prototype system and algorithm to answer nearest neighbor queries over objects moving in road networks. Papadias, Zhang, Mamoulis and Tao integrated network and Euclidean information to efficiently prune the search space and answer range, nearest neighbor, closest pairs and e-distance join queries in the context of spatial network databases [10]. In [5], Kolahdouzan and Shahabi proposed a novel approach to efficiently and accurately evaluate KNN queries in spatial network databases using first order Voronoi diagram. This approach is based on partitioning a large network to small Voronoi regions, and then pre-computing distances both within and across the regions. Shahabi proposed an embedding technique that approximates the network distance with computationally simple functions in order to retrieve fast, but approximate, k-NN results [12]. The Incremental Monitoring Algorithm (IMA) and Group Monitoring Algorithm (GMA) algorithms [9] are introduced to calculate continuous nearest neighbor in **road networks**. IMA retrieves the initial result of a query q by expanding the network around it until k NN's are found. GMA benefits from the shared execution among queries in the same path, and the reduction of the problem from monitoring moving queries to (monitoring) static network nodes. PLACE* [13] introduces Query-Track-Participate (QTP) processing model to process continuous queries. Figure 1 shows the steps of query evaluation in PLACE*. In PLACE*, each moving object is associated with a server called its visiting server ($VS(O)$, initially it is object's home server $HS(O)$). For a query q, the querying server $QS(q)$ is the regional server of that the query issuer, i.e., $QS(q) = VS(i_q)$. A participating server for a query q, PS(q), is a regional server whose coverage region overlaps the search region of q. For a query q, the tracking server $TS(q)$ is the regional server that q's focal object, f_q, currently belongs to. Participating servers send update of the query result to the query server QS of the query. The QS forwards updates to the query issuer (i). If the focal of the query changed its position, $TS(q)$ sends the new position to $QS(q)$. $QS(q)$ calculates the new search region for q and updates the participating servers.

Sallam used an enhanced version of IMA, for query processing over road networks, and applied it on PLACE* QTP distributed architecture [11]. In our work, We applied Sallam's algorithms in [11] for continuous queries over **road networks** on our MobiPLACE* system architecture. We aim to provide more system scalability by enhancing server response time and reducing memory usage. A comparison between Sallam's algorithms performance on PLACE* and

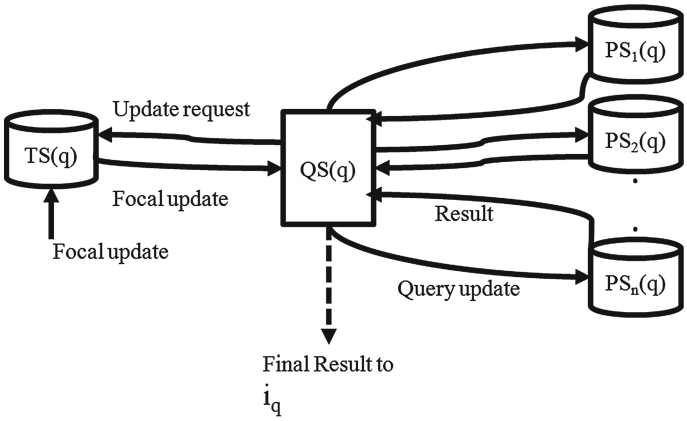

Fig. 1. Query evaluation in PLACE*

on MobiPLACE* architectures is made to illustrate the benefit of using MobiPLACE* architecture.

3 MobiPLACE*: An Overview

In this section, we discuss the details of MobiPLACE* architecture and algorithms to process continuous queries. Query processing algorithms are introduced in [11] with different roles of system components. We modified those algorithms to work correctly with our architecture. In Sect. 3.1, we discuss the details of system architecture and the role of system components. In Sect. 3.2, we present the communication messages between system components in order to execute queries, update object location and update query result.

3.1 System Architecture

MobiPLACE* system architecture is inspired from Query-Track-Participant (QTP) communication model introduced in PLACE* [13]. We divided the area where objects are moving into regions and each region is covered by a regional server. In PLACE*, each regional server has 3 roles; querying, tracking and participating server roles. In MobiPLACE*, the role of querying is moved from server to mobile client. When clients connect to our system, a mapping file is downloaded to those clients. This mapping file determines the IP address of each regional server paired with its coverage region boundaries. When a client needs to initiate a query, it connects to the home server of this query focal ($HS(f_q)$) to know its current regional server which will act as a tracking server to the query. For simplicity, we assume that each object is the focal of its queries and in this case the previous step could be ignored. Then the client determines the set of regional server to participate on this query. It connects to them and gather the

result. The result is summarized on the client. This will reduce the processing and memory requirements on server. Now, each client device will be responsible on handling its own query instead of depending on server to handle all steps of queries from all objects on the system. Figure 2 illustrates the architecture of MobiPLACE* system.

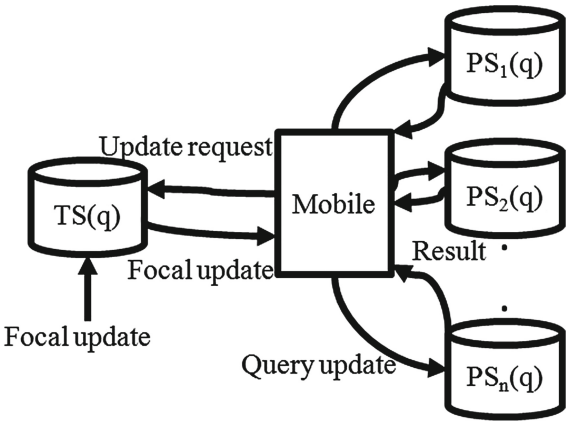

Fig. 2. MobiPLACE* system architecture

3.2 Communication Messages

The area is divided into n sub-regions that are covered with n regional servers; each is responsible on a sub-region. Information of road junctions and roads is stored at servers responsible on this region. All-pairs shortest paths, between road junctions, matrix is pre-computed. Each server stores the shortest path cost from all nodes in the network to the junctions it stores in a list sorted according to the path cost and this will be used in query evaluation. When a new object (O) connects to the system, the following steps are taken as below.

1. Object O sends a connection request to the Default Server DS (a previously known server for all objects upon installing our application on O).
2. DS searches the regional servers and locate O's Home Server (HS(O)) which covers O's location and sends its connection details to O.
3. O sends a connection message to HS(O).
4. HS(O) attaches O to the nearest road to its location and generate a unique identifier for the object O by adding a prefix of server's ID to object O ID (will be used later to locate HS(O)).
5. O sends updates to its home server until it moves outside its coverage area and in this case O migrates to another server which is called O's visiting server VS(O). HS(O) keeps track of O's current visiting server for easier locating later.

Moving objects can issue range and k-NN queries. As a difference from PLACE*, Querying Server role is now moved on mobile client. This means that mobile client and participating servers cooperate to answer the query continuously.

The steps of query execution on MobiPLACE* architecture are similar to those on PLACE* architecture except moving the querying server execution steps on mobile devices. In order to enable this, a mapping file of region/IP of regional server (which tells the mobile client the coverage area of each server and its communication information) is downloaded on new object connection to our system. When a query initiated on the mobile client, the issuer i asks the home server of query focal f (this is known from object's ID prefix) about the current visiting server of the focal VS(f) which will act as a tracking server for this query TS(q). The home server informs the issuer about the current TS(q) of its query. The issuer then contacts the TS(q) to know the exact location of the query focal f. Using the regional server mapping file, the issuer i can determine the set of participant servers PSs(q) for this query that will participate to collect the query result. TS(q) informs the issuer of any updates of focal location and based on those update, the issuer updates the plan and informs the participants the new plan. Without loss of generality, we can assume that the issuer itself is the query focal. In this case, the steps of asking about TS can be ignored.

When an object O issues a new query q (assuming that the issuer itself is the query focal), the following steps describes how to evaluate range and k-NN queries.

1. O expands the search from its position.
2. O finds the regional servers whose regions overlap the query search region PS(q).
3. O sends an evaluation request to all PS(q). This request contains O's position and query parameters (i.e. range in range queries) and waits for answers from PSs(q). O keeps track of its queries that have not been answered completely yet by keeping track which servers have responded and which have not yet.
4. When receiving the answer from all servers in PSs(q), O gather the result and display it as the final result to the user. O can display results incrementally upon receiving any answer from any server but making sure that those answers should be updated and may content some false results (i.e. in k-NN query).

3.3 Continuous Range Query Evaluation

To evaluate a new range query q, the visiting server of the focal start expanding from focal's road ends to the neighbor road junctions. It either stops when reaching nodes out of query's range or reaching some road junctions covered by another server. At the same time, each participant server start a similar process (expanding) like the visiting server of the focal but from road junctions that are within query range. Server can know which nodes are in range using all-pair shortest path matrix, which is previously calculated offline, between road junctions. Those junctions are pushed in a queue to continue the expansion process

using similar algorithm in [11]. We define a leaf junction in 2 ways. Either it has only one neighbor junction and we could not expand from it anymore or it is out of range junction that is a neighbor to an in range junction. Those leaves are used in incremental evaluation of queries as described later in Sect. 3.5.

3.4 Continuous k-NN Query Evaluation

The range of k-NN query is not known in advance. By assuming uniformity of object distribution over servers, we can transform k-NN query to a range one by estimating the range to find k neighbors. As in [13], we can calculate the range of objects within object's regional server by $d * \sqrt{k/n}$ where d^2 is the area covered by objects' server and n is the number of objects in this server (Fig. 3). After calculating the range of the query, requests to PS are sent and results are gathered and sorted in a min priority queue. If the result reaches k objects or more, the algorithm is stopped and result is displayed. If not, the range is expanded by a factor and the process is repeated. Server expand the search of the query by starting from leaves junctions and start expanding to the new range. Results are updated to client until reaching k moving objects. Participant servers of the query store the leaves of the search tree in order to be used later in the incremental updates.

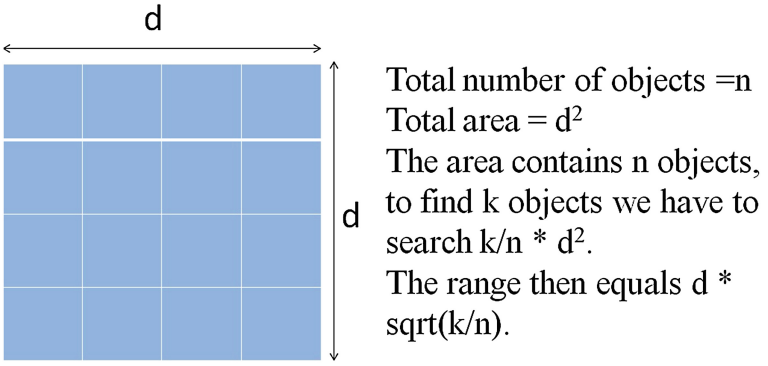

Fig. 3. Calculate k-NN query's range

3.5 Query Incremental Evaluation

Every object periodically updates its location to its current visiting server VS(O). If O moves outside its current visiting server, it sends a connection request to the new Visiting Server and a disconnection message from to the old one. It will also notify its Home Server with the new visiting server ID.

Upon receiving an update from an object O, VS(O) performs the following steps.

1. Locate the new road for O and calculate the cost from O to this road end junctions.
2. Update mobile clients whose queries are affected either by adding or removing O.
3. Send message to O itself confirming its connection to VS(O).

4 Performance Evaluation

Amazon instances with dual 1.88 GHZ processors and 1.7 GB of RAM are used in the experiments. The region under studying is divided into 4 equal sub-regions. A regional server is responsible for each region. Each regional server runs on a dedicated machine. Servers are connected on Amazon private network and TCP connections are used as a connection protocol. Two more servers are used as a default server and an event simulator server. There are many input parameters to the simulation model: road network, represented by the set of nodes and edges, moving objects number, objects' velocity and update period. We used the city of Oldenburg in Germany as the underlying network with 6105 road junctions and 7035 edges. Input parameters are summarized in Table 1.

Table 1. Summary of system input parameters

Parameter	Range values	Default value
Network edges	–	7035
Network nodes	–	6105
Update period	–	10 s
Moving objects velocity	Low, medium, high	Medium (50 Km/h)
Object update percentage	10, 50, 100 %	10 %
k of k-NN	1, 10, 100, 1000	100
R of range	2, 5, 7, 10, 30 (%)	10 %
Population size	50K, 100K	50K

We used Thomas Brinkhoff [1] generator to generate 50K moving objects over road networks. Objects update their position every 10 s and randomly generate continuous queries. We made many experiments to compare between MobiPLACE* and PLACE* processing models. We focused on response time and server memory usage in the comparison.

Figures 4 and 5 show the effect of varying the parameters of range and k-NN queries respectively on response time of the proposed two architectures. Figure 4 studies the effect of changing the range between 2 %, 5 %, 7 %, 10 % and 30 % of the network area when the population size is 50K moving objects, while Fig. 5 does the same with NN queries. The figures show that MobiPLACE* with client connection bandwidth equals 1 Mb/s performs better than PLACE*, on the same

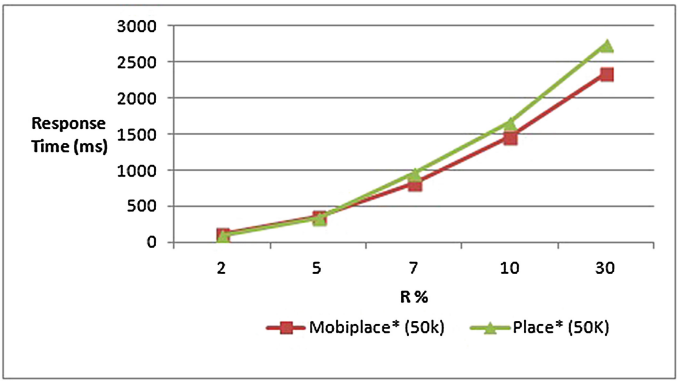

Fig. 4. Range query response time with 50K population

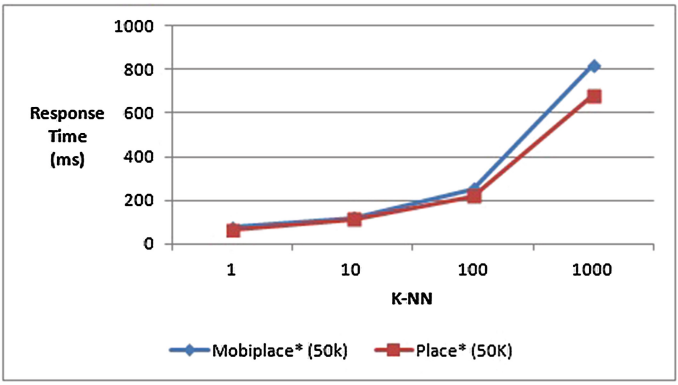

Fig. 5. k-NN response time with 50K population

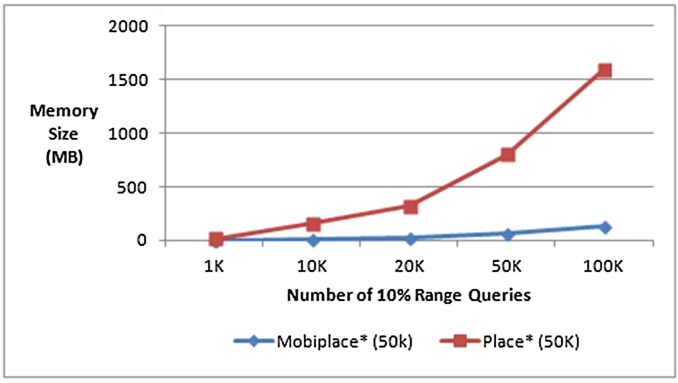

Fig. 6. Memory requirements for 10 % range queries

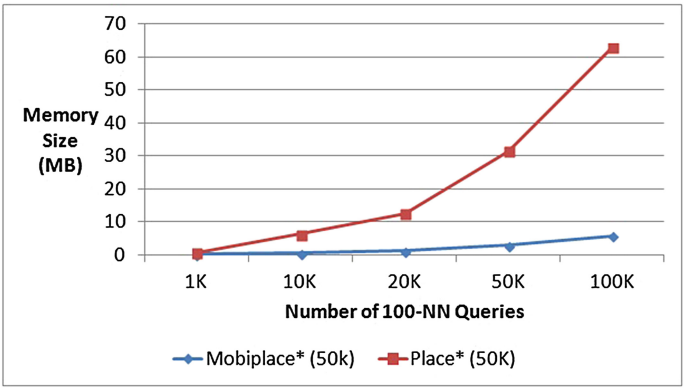

Fig. 7. Memory requirements for 100-NN queries

algorithms and conditions, when query range becomes large and show than it is a little worth in case of k-NN because the amount of data sent to client object is larger in case of MobiPLACE*.

Figure 6 shows the estimated total server memory usage of range queries on the 2 architectures and verifies that MobiPLACE* requires a significant lower amount of memory than PLACE*. Figure 7 shows the same for k-NN queries.

5 Conclusion

In this paper, we discussed MobiPLACE* a distributed framework for continuous processing of spatio-temporal queries over road network by utilizing mobile clients' processing power. It is built based on PLACE* QTP communication model. Experiments showed that moving Query Server role to mobile clients slightly enhanced query response time and significantly reduced the server memory usage. We have many future extension that we could not cover in this paper. Many papers, that solved the same problem, did not mention how to distribute regional servers over region. We need to determine a set of standard experiments to calculate the best number of servers and their distribution to achieve the best query's average response time for each application. Privacy issues should be taken into consideration. Also, we need to make a detailed study of the communication links effect on performance. We took connection bandwidth as a parameter and simulated it only using delays. More connection details like latency, network congestion and connection initiation time should be taken into consideration later in order to provide more accurate results.

References

1. Brinkhoff, T.: A framework for generating network-based moving objects. GeoInformatica **6**(2), 153–180 (2002)
2. Cai, Y., Hua, K.A., Cao, G.: Processing range-monitoring queries on heterogeneous mobile objects. In: Proceedings of the 2004 IEEE International Conference on Mobile Data Management, 2004, pp. 27–38. IEEE (2004)
3. Gedik, B., Liu, L.: MobiEyes: distributed processing of continuously moving queries on moving objects in a mobile system. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 67–87. Springer, Heidelberg (2004)
4. Jensen, C.S., Kolářvr, J., Pedersen, T.B., Timko, I.: Nearest neighbor queries in road networks. In: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems, pp. 1–8. ACM (2003)
5. Kolahdouzan, M., Shahabi, C.: Voronoi-based k nearest neighbor search for spatial network databases. In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, vol. 30, pp. 840–851. VLDB Endowment (2004)
6. Mokbel, M.F., Aref, W.G.: SOLE: scalable on-line execution of continuous queries on spatio-temporal data streams. VLDB J. **17**(5), 971–995 (2008)
7. Mokbel, M.F., Xiong, X., Aref, W.G.: SINA: scalable incremental processing of continuous queries in spatio-temporal databases. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 623–634. ACM (2004)
8. Mouratidis, K., Papadias, D., Hadjieleftheriou, M.: Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 634–645. ACM (2005)
9. Mouratidis, K., Yiu, M.L., Papadias, D., Mamoulis, N.: Continuous nearest neighbor monitoring in road networks. In: Proceedings of the 32nd International Conference on Very Large Data Bases, pp. 43–54. VLDB Endowment (2006)
10. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: Proceedings of the 29th International Conference on Very Large Data Bases, vol. 29, pp. 802–813. VLDB Endowment (2003)
11. Sallam, A., Nagi, K., Abougabal, M., Aref, W.G.: Distributed processing of continuous spatiotemporal queries over road networks. Alex. Eng. J. **51**(2), 69–152 (2012)
12. Shahabi, C., Kolahdouzan, M.R., Sharifzadeh, M.: A road network embedding technique for k-nearest neighbor search in moving object databases. GeoInformatica **7**(3), 255–273 (2003)
13. Xiong, X., Elmongui, H.G., Chai, X., Aref, W.G.: PLACE*: a distributed spatiotemporal data stream management system for moving objects. In: 2007 International Conference on Mobile Data Management, pp. 44–51. IEEE (2007)
14. Xiong, X., Mokbel, M.F., Aref, W.G.: SEA-CNN: scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In: Proceedings of the 21st International Conference on Data Engineering, 2005. ICDE 2005, pp. 643–654. IEEE (2005)
15. Yu, X., Pu, K.Q., Koudas, N.: Monitoring k-nearest neighbor queries over moving objects. In: Proceedings of the 21st International Conference on Data Engineering, 2005. ICDE 2005, pp. 631–642. IEEE (2005)