Distributed Processing of Continuous Range Queries Over Moving Objects

Jin Zhou, Hao Teng, Ziqiang Yu(✉), Dong Wang, and Jiaqi Wang

Shandong Provincial Key Laboratory of Network Based Intelligent Computing, University of Jinan, Jinan 250022, China
ise_yuzq@ujn.edu.cn

Abstract. With the widespread usage of wireless network and mobile devices, the scale of spatial-temporal data is dramatically increasing and a good deal of real world applications can be formulated as processing continuous queries over moving objects. Most existing works investigating this problem mainly concern about the centralized search algorithm for dealing with range queries over a limited volume of objects, but these approaches hardly can scale well in a cluster of servers. Additionally, the existing approaches seldom process the situation that the locations of objects and queries are simultaneously changing. To address this challenge, we propose a distributed grid index and a distributed incremental search approach to handle concurrent continuous range queries over an ocean of moving objects. As to the distributed grid index, it can be deployed on a distributed computing framework to well support the real-time maintenance of moving objects. Further, we take fully into account the condition that locations of objects and queries are both changing at the same time, and put forward a parallel search approach based on the publish/subscribe mechanism to achieve incrementally searching results of each continuous range queries with a cluster of servers. Finally, we conduct extensive experiments to sufficiently evaluate the performance of our proposal.

Keywords: Continuous range queries · Distributed processing · Incremental search · Moving objects

1 Introduction

With the popularity of mobile devices and wireless network, many things such as sharing bicycles can be deemed as moving objects. The problem of processing continuous range queries over moving objects has attracted extensive attentions because many location based services can be formulated as this problem. The semantic of the continuous range query in our work refers to the locations of queries and objects are both continuously changing. Although this semantic enhances the difficulty of processing this type of queries but it accords with the reality and there are some cases that can illustrate this point. To capture suspects, polices will monitor the vehicles passing into or out of a specified region and the polices will probably adjust the search region frequently, which

D.-S. Huang et al. (Eds.): ICIC 2017, Part II, LNCS 10362, pp. 800–810, 2017.
DOI: 10.1007/978-3-319-63312-1_72

is indeed a continuous query processing because the results need to be consecutively updated with vehicles moving and the query scope is also changing constantly.

In another example, a comprehensive service in cap-hailing applications is that the data center needs to continuously seek for nearby taxies for a user who is walking, and this service is also a continuous range query with locations of queries and objects are all dynamically varying. Since the continuous range query has broad applications, so we concentrate on devising an distributed framework with efficiency to address this problem.

In this work, a continuous range query (*CRQ* for short) over moving objects refers to returning moving objects inside a user-defined region in real-time and continuously monitoring the change of query results as the query region constantly changes over a certain time period. In the big data background, processing *CRQ*s over moving objects is faced with unprecedented challenges. First, we have to face the tremendous volumes of moving objects and queries, which are far beyond the computing and storage capacities of one single server. Second, with the ubiquitous mobile internet, most range queries are online and they desire to be responded in real-time. Last but not the least, it is necessary to constantly monitor the results of *CRQ*s as objects and queries are all moving, which probably involves a gigantic computing cost.

For the sake of challenges, most of existing works are not suitable for dealing with the continuous range queries. This is because many works [7, 10] do not consider the situation that locations of queries and objects are both changing simultaneously. Another critical issue is that most existing proposals [2, 4] always investigate the central algorithms to improve the search efficiency, but they cannot scale well to handle extensive concurrent range queries over a tremendous set of moving objects. Due to the limitations of existing methods, this work explores a distributed framework to search and monitor the results of given continuous range queries based on a cluster of servers in real-time. Specifically, we construct a distributed grid index structure that can be seamlessly deployed in a master-slaves model to support the maintenance of moving objects and the parallel processing of range queries. We further design a distributed incremental search method that can update the results of each *CRQ* with only computing the incremental result.

Our main contributions can be summarized as follows:

- This work process *CRQ*s in the scenario where queries and objects are simultaneously moving, which is seldom involved in other existing works.
- We propose DGI, a distributed grid index, for supporting *CRQ*s over moving objects in a distributed setting.
- We design DIS, a comprehensive distributed incremental search approach that can take full advantages of a cluster of servers to continuously monitor the results of *CRQ*s in real-time.
- DGI and DIS are implemented on top of S4, and we conduct extensive experiments to evaluate the performance of DGI and DIS, which confirm its superiority over existing approaches.

2 Related Work

Continuous range query processing is very important due to its broad application base, and it has been extensively studied by existing proposals. Some works [7, 9] investigate processing range queries on road network. Stojanovic et al. [7] propose a framework for continuous range query processing for objects moving on network paths, and introduces an additional pre-refinement step which generates main-memory data structures to support reevaluation of continuous range queries. The work [9] studies the problem of processing range queries on road networks and proposes Voronoi Range Search based on the Voronoi diagram. Due to determining the Voronoi diagram for each object is very expensive, so this approach hardly can support range queries on frequent moving objects. Additionally, some works introduce the concept of safe region to reduce the reevaluation cost. The proposal [4] points out that the cost of monitoring and keeping the location of a moving query updated is very high, and it investigates an efficient technique by adopting the concept of a safe region. That is, as long as the query remains inside its specified safe region, expensive re-computation is not required.

Cheema et al. [2] also adopt the methodology that utilizes he concept of a safe zone to monitor moving circular range queries and propose powerful pruning rules improve the query efficiency. The above works all focus on searching the exact results of queries, but the work [5] pay more attention to approximate range search and proposes approximate static range search (ARS). The work [8] coins a term "Region Queries" to indicate a broad category of spatial range queries, and focuses on showing a complete picture of region queries. These proposals present some excellent search algorithms for monitoring (continuous) range queries, but methods are centralized and their scalability is restricted. Moreover, most of them does not consider the situation that every object is constantly moving as processing continuous range queries.

It is imperative that utilizing the distributed computing model to deal with concurrent range queries over moving objects, and some works [1, 3, 6, 10, 11] have explored this problem. In fact, the distributed framework in these approaches consist of a central server and extensive moving objects and they all require the moving devices to have considerable computational capabilities, which restricts their applicability. In contrast, our approach does not assume any computation capabilities at the mobile objects other than reporting their positions (e.g., the sharing bicycles can be a simple GPS tracking device), and thus has wider applicability.

3 Distributed Grid Index

Given an interest of region covering a large number of moving objects, a grid index partitions this region into four-square cells with the same size. For any cell c_i, it is regarded as an index unit that records the locations of moving objects residing in the cell as well as the queries involving it. Grid index structure has been extensively utilized for processing spatial-temporary queries, and we also utilize it to support the processing of CRQs over moving objects. But unlike the existing approaches utilizing the grid index on a single server, we construct a Distributed Grid Index (DGI), namely, deploying the

grid index on a master-slaves model, which consists of one master and multiple servers. Next, we will expound the structure of DGI as well as the deployment of DGI on the master-salves model. The symbols will be used in later text is summarized in Table 1.

Table 1. Summary of symbols

Symbols	Meanings
$o \prec B$	An object o is covered by the region B
$o \nprec B$	An object o is not covered by the region B
$C \sqcap D$	The region C is partially covered by the region D
$C \overline{\sqcap} D$	The region C is fully covered by the region D
H \ Z	The elements belong to t H but do not belong to Z

DGI consists of a Global Schema Index (GSI) and extensive cell indexes. In particular, DGI refers to deploying the grid index on a *master-salves* cluster with a general streaming data processing model. In this model, we use the conception of PE to represent a logical processing element, and the data will be encapsulated as an *event* that can be transferred from one PE to another. Every *event* is formulated as a triple (*type*, *key*, *value*). When a PE consumes a received *event*, it will encapsulate the intermediate results as a new event that can be routed to another PE. Here, each PE can receive their desired *events* by specifying their *type*s and *key*s, which indeed forms the *events* routing rule. When this model assigns an *event* to a corresponding PE based on the routing rule, if the PE does not exist, a new PE will be automatically created to process this *event*.

In DGI framework, a unique EntrancePE on the *master* maintains the GSI and every CellPE takes charge of one or more cell indexes in different slaves. GSI is maintained by the *master* and cell indexes are distributed in different *slaves*. For GSI, it needs to contain identifiers and boundaries of every cell index. In addition to this, it also has to *master* the relationships between cells and *slaves*. To achieve above purposes, the GSI seems to be necessary to store much information that is apt to make it be a bottleneck. But in fact, the GSI designed by us only needs to record the bottom left cell as the reference cell and a naming rule that can be used to identify the identifier of each cell rapidly. Since every cell is a four-square, then the identifier and boundaries of every cell can be deduced instantly. As to the relationships between cells and *slaves*, we introduce an hash function *f(x)* that can map the cells to different slaves on the basis of the identifiers of cells, that is, $s_i = f(c_i)$, where s_i and c_i are the identifiers of slaves and cells.

Each cell index is charged of recording the objects covered by itself and the queries with search scopes intersecting with this cell. For any cell c_i, it has three major components, i.e., OL_i, FL_i, and PL_i. The list OL_i is used to record the locations of moving objects covered by itself. FL_i stores the identifiers of queries whose search regions fully cover c_i, while the queries with the search scope partially overlapping with c_i are maintained by PL_i.

4 Distributed Incremental Search Approach

In this section, we propose DIS, a comprehensive approach to address the challenges of incrementally searching the results of extensive *CRQ*s in real-time.

4.1 Search Initial Results of *CRQ*s

The DRQS framework (as shown in Fig. 1) can be deemed as the fundamental computing architecture, based on which we design search algorithms to process *CRQ*s. Now we discuss SIR, an algorithm to search initial results of *CRQ*s.

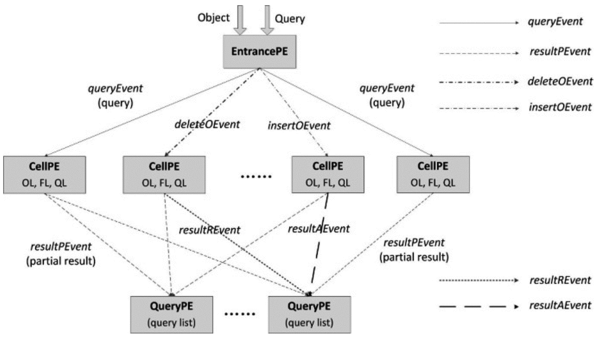

Fig. 1. The framework of DRQS

When EntracePE receives a *CRQ* (q_i) with the search scope sq_i, it first determines the cells intersecting with sq_i based on *GSI* and these cells are called *candidate cells*. We use $\mathcal{F}$ to label the set of *candidate cells*. Since the boundaries of cells are static, the cells in $\mathcal{F}$ can be determined with a brute-force comparison of boundaries of each cell with sq_i, but which will greatly waste computing costs. To settle this issue, we will firstly find the cell c_h that covers the center point of the search scope sq_i, and then detect whether each adjacent cell c_i of c_h has an intersection with sq_i. This process will iteratively enlarge the search region until a group of cells surrounding c_h do not overlap with sq_i. This strategy can quickly find the candidate cells of q_i even if the shape of sq_i is irregular.

After determining the set $\mathcal{F}$ for q, EntrancePE will send a *queryEvent* carrying q to the CellPEs corresponding to cells in $\mathcal{F}$. To facility presentation, we suppose that each CellPE is charged of one cell index. Hence, when a CellPE receives the *queryEvent*, it will instantly find the objects covered by sq_i and these objects form a partial result of q_i. Every CellPE encapsulates the partial result as a *resultPEvent* and sends it to the *Query*PE. In DRQS framework, the final result of every query is calculated and maintained by a unique QueryPE, and the *resultPEvent* carrying partial results of q_i will be routed to the same QueryPE with identifier of q_i as the key of *resultPEvent*. Therefore, QueryPE can obtain the final result of q_i by merging all partial results.

4.2 Incrementally Computing Results of *CRQ*s

After obtaining the initial results of *CRQ*s, we still need to monitor the results of every query. We first design IOS, an algorithm for incrementally searching results of *CRQ*s as object are moving based on a publish/subscribe mechanism to constantly update the results for each query in real-time. In IOS algorithm, every CellPE and QueryPE are regarded as a data publisher and query subscriber respectively. We use cp_i to label the CellPE matching the cell c_i. As to the CellPE cp_i, it maintains two registered query lists FL_i and PL_i. For a given query q_i, its candidate cells forms the set $\mathcal{F}$. If $c_i \in \mathcal{F}$, q_i will be registered in c_i. Specifically, q_i will be inserted into FL_i if $sq_i \simeq c_i$ or inserted into PL_i if $s_q \sim c_i$. The query q_i has to be registered in every candidate cell in this way. Once an object in the cell c_i moves, the corresponding CellPE only needs to detect whether the registered queries can be influenced rather than recompute the results of all existed queries.

In fact, we further observe that not all registered queries in c_i will be influenced by the movement of every object. Since the result of q_i just concerns about the objects covered by sq_i rather than their exact positions, so we can deduce that if a CRQ covers the cell ci, it will not be influenced by the movements of objects in

Theorem 1. *For a given cell* c_i, $\forall\, o_i \in OL_i$, *its location is* $\left(x'_i,\ y'_i\right)$ *at the time point* t_i *and the location becomes* (x_i, y_i) *at the time point* t_{i+1}. *If* $\left(x'_i,\ y'_i\right)$ *and* (x_i, y_i) *are both covered by* c_i, *then* $\forall\, q_j \in FL_i$, *its result will not be influenced by the changed location of* o_i *with the condition that* $t^s_{q_j}$ *is smaller than* t_i.

With the help of Theorem 1, if the location of o_i in a cell c_i changes, we should update the results of the registered queries in c_i by handling the following two cases.

- The first case is that $\left(x'_i,\ y'_i\right) \prec c_i$ and $(x_i, y_i) \prec c_j$ $(i \neq j)$. In this case, the results of all registered queries of c_i and c_j will be probably influenced. For each registered query q_i of c_i, if $\left(x'_i,\ y'_i\right) \prec sq_i$, then the CellPE corresponding to c_i will notify the QueryPE maintaining q_i to remove $\left(x'_i,\ y'_i\right)$ from the result of q_i by sending a *resultREvent*; meanwhile, for every registered query q_j of c_j, if $(x_i, y_i) \prec sq_j$, then the CellPE matching c_j will send a *resultAEvent* to the QueryPE maintaining q*i*, which aims to notify this QueryPE to insert o_i (x_i, y_i) into results of q_j.
- Another case is that $\left(x'_i,\ y'_i\right) \prec c_i$ and $(x_i, y_i) \prec c_i$. According to Theorem 1, $\forall\, q_i \in FL_i$, it cannot be affected by the object o_i. Hence, only the results of queries in PL_i need to be updated. In this case, the CellPE matching c_i can directly identify the registered queries affected by oi, and then notify the corresponding QueryPEs to update the query results.

As above, every CellPE corresponds to a publisher and each QueryPE serves as a subscriber. Only if a *CRQ q* is registered in the CellPEs corresponding to its candidate

cells, these CellPEs will continuously monitor the results of q in a parallel way only if their matching cells belong to the set $\mathscr{F}$ of q. In this process, different CellPEs will send the moving objects involved with q to a unique QueryPE, which can be deemed a subscriber. This QueryPE will process the received locations of objects as soon as possible to guarantee the exact result of q all the time. Moreover, the QueryPE as a subscriber will take charge of maintaining the result of the query q all through the life-cycle of q. In this scenario, we organize extensive CellPEs and QueryPEs in different servers as a publish/subscribe mechanism that can well support incrementally searching the results of *CRQ*s in a distributed environment.

As to a query q_i, if its search scope is sq_i at time point t_j and sq_i becomes sq_i' at time point t_{j+1}, then q_i is required to be resubmitted to EntrancePE. When receiving q_i, EntrancePE will compute $\mathscr{F}$, the set of candidate cells for q_i, based on sq_i', and then send q_i to each cell c_k ($c_k \in \mathscr{F}$). After receiving q_i, the cell c_k has to update its two lists of registered queries. Meanwhile, the matching CellPE will instantly find the objects covered by sq'_i from the cell c_k. In this case, CellPE indeed employs an incremental search strategy to only compute the incremental result of q_i at time point t_{j+1} based on its existing result, and this incremental search strategy includes the following steps.

(1) If q_i is a new registered query for c_k, the following cases need to be considered.

- If $sq_i' \simeq c_k$, then q_i will be inserted into FL_k and all objects of c_k form a part of the result of q_i.
- If $sq_i' \sim c_k$, then q_i will be inserted into PL_k and c_k has to find the objects covered by sq'_i.

(2) If q_i is an existed registered query of c_k, it will be handled with next steps.

- If $(sq_i \simeq c_k)$ && $(sq_i' \simeq c_k)$, which means q_i has been in FL_k and we have no need to insert q_i into FL_k again. In this case, q_i still covers all objects in c_k though it moves. As to the cell c_k, it does not need to update the result of q_i;
- If $(sq_i \simeq c_k)$ && $(sq_i' \simeq c_k)$, we need to remove q_i from PL_k and insert it into QL_k. In this case, the incremental result of q_i is the set of objects covered by the region $\left(sq_i' - sq_i\right)$, which can be rapidly determined.
- If $(sq_i \simeq c_k)$ && $(sq_i' \sim c_k)$, we has to remove q_i from QL_k and insert it into PL_k. At this moment, we only need to remove the objects residing in the scope $\left(sq_i - sq_i'\right)$ from the result of q_i.
- If $(sq_i \sim c_k)$ && $(sq_i' \sim c_k)$, q_i is not necessary to be added into PL_k again but $c_k$needs to update the information of q_i. Now, we need to search all objects covered by the region $(sq_i' - (sq_i' n \bigcap sq_i))$ and these objects belong to the new result of q_i.

5 Experiments

We conduct experiments to evaluate the proposed DGI index and DSI approach. To better evaluate the performance of DSI, we introduce two other distributed algorithms as baseline methods. The first method, NS, is a naive search algorithm which does not use any index. For any object, NS uses a hash function to determine which server should store it. Processing a *CRQ* thus involves scanning all objects maintained in all servers at any time point. The second method, S-DIS, is a simplified DIS method that handles a *CRQ* as a new query at any time point, that is, it does not utilize the incremental search strategy when processing *CRQ*s.

We use the German road network data to simulate two different datasets for our experiments. In the datasets, all objects appear on the roads only. In the first dataset (UD), the objects follow a uniform distribution. In the second dataset (GD), 70% of the objects follow the Gaussian distribution, and the other objects are uniformly distributed. In these two datasets, the whole area is normalized to a unit square and this square is partitioned into small cells with edge length being 0.01. Moreover, all objects move along the road network, with the velocity uniformly distributed in [0, 0.002] unless otherwise specified. We use V_p and V_q to represent the velocities of an object and a query. The experiments are conducted on a cluster of 8 Dell R210 servers with Gigabit Ethernet interconnect.

We first evaluate the performance of DGI. Figure 2 demonstrates the time of building DGI with the number of objects varying. Based on the results, we observe that the time cost is in proportion to the number of objects and it is almost not influenced by the distribution of objects. When objects are moving, the velocity of objects will exert an impact on the build time of DGI. In Fig. 3, we evaluate the maintaining time of DGI by processing a set of objects in a specified period of time and find that the maintaining time is slightly affected by the velocity of objects. This is because DGI always needs to remove the obsolete location and insert the new position for processing the movement of an object regardless of the distance it moves one time.

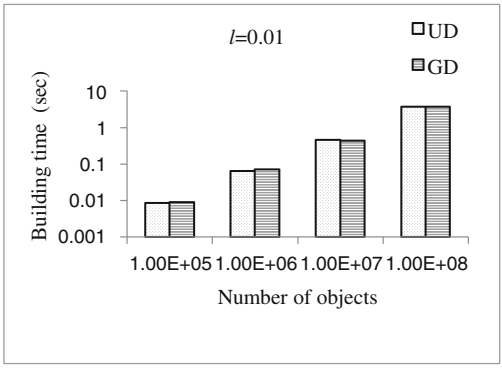

Fig. 2. Building time of DGI

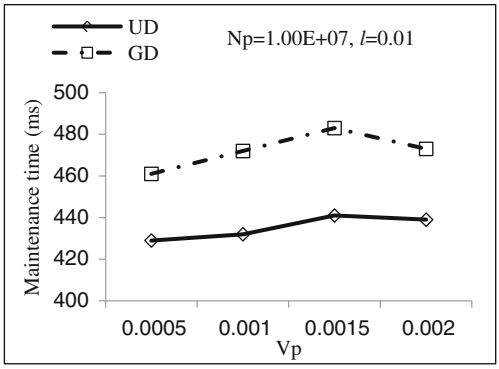

Fig. 3. Influence of V_p on maintenance time of DGI

Next, we conduct an evaluation on the performance of DIS approach. In Fig. 4, we first test the time of three approaches when processing the same set of queries. In this group of experiments, we make 10000 queries continuously move and test the time of processing these queries at five consecutive time points. As can be observed, DIS performs better than other two approaches especially at the last four time points. The reason why the time cost of DIS decreases is that only incremental result of queries need to be calculated after the first time point, which can greatly reduce the computing time, while other two methods always process each query as a new one.

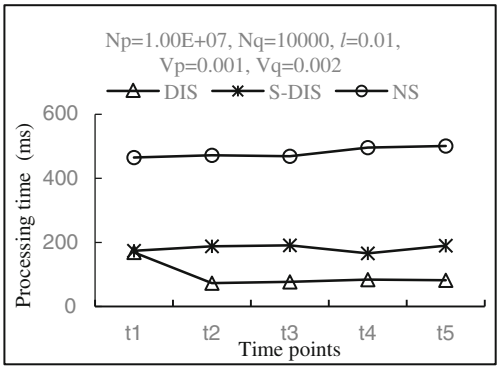

Fig. 4. Comparison of three approaches

Figure 5 demonstrates the influence of number of queries on the processing time of each method. Here, we observe that the response time of each approach increases obviously as more and more queries are processed, but the consuming time of NS approach grows more sharply. Due to this set of experiments does not involve continuous queries, so the performances of DIS and S-DIS are almost identical.

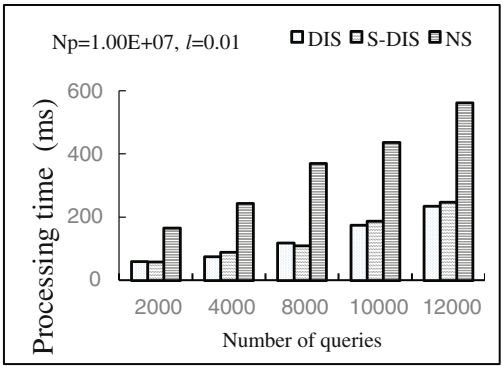

Fig. 5. Processing time of three approaches

6 Conclusion

With the dramatic increase of mobile devices and the advances in wireless network, the efficient processing of *CRQ*s has been of increasing interest. This work propose a distributed incremental search approach that sufficiently considers the situation that queries and objects are both moving and only needs to reevaluate the incremental result of every *CRQ* to cut down the computing costs as well as communication expenses between CellPEs and QueryPEs. Finally extensive experiments are conducted to verify the performance of our proposal.

Acknowledgement. This work was supported in part by the Shandong Provincial Natural Science Foundation (ZR2016FB14) Science and Technology Program of University of Jinan (XKY1737), the National Natural Science Foundation of China under Grants with No. 61640218, and the Project of Shandong Province Higher Educational Science and Technology Program under Grant with No. J16LN07.

References

1. Cai, Y., Hua, K.A., Cao, G.: Processing range-monitoring queries on heterogeneous mobile objects. In: 2004 IEEE International Conference on Mobile Data Management, pp. 27–38 (2004)
2. Cheema, M.A., Brankovic, L., Lin, X., Zhang, W., Wang, W.: Multi-guarded safe zone: An effective technique to monitor moving circular range queries. In: ICDE 2010, pp. 189–200 (2010)
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., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 67–87. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24741-8_6
4. Haidar, A.-K., Taniar, D., Betts, J., Alamri, S.: On finding safe regions for moving range queries. Math. Comput. Model. **58**(5), 1449–1458 (2013)

5. Haidar, A.-K., Taniar, D., Safar, M.: Approximate algorithms for static and continuous range queries in mobile navigation. Computing **95**(10–11), 949–976 (2013)
6. Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 479–490 (2005)
7. Stojanovic, D., Papadopoulos, A.N., Predic, B., Djordjevic-Kajan, S., Nanopoulos, A.: Continuous range monitoring of mobile objects in road networks. Data Knowl. Eng. **64**(1), 77–100 (2008)
8. Taniar, D., Rahayu, W.: A taxonomy for region queries in spatial databases. J. Comput. Syst. Sci. **81**(8), 1508–1531 (2015)
9. Xuan, K., Zhao, G., Taniar, D., Rahayu, W., Safar, M., Srinivasan, B.: Voronoi-based range and continuous range query processing in mobile databases. J. Comput. Syst. Sci. **77**(4), 637–651 (2011)
10. Zhou, J., Chen, L., Chen, C.L.P., Zhang, Y., Li, H.: Fuzzy clustering with the entropy of attribute weights. Neurocomputing **198**, 125–134 (2016)
11. Zhou, J., Chen, C.L.P., Chen, L., Li, H.: A collaborative fuzzy clustering algorithm in distributed network environments. IEEE Trans. Fuzzy Syst. **22**(6), 1443–1456 (2014)