2015 16th IEEE International Conference on Mobile Data Management

A Framework for Spatial Predictive Query Processing and Visualization

Abdeltawab M. Hendawi[1,2] Mohamed Ali[2] Mohamed F. Mokbel[1*]

[1]*Department of Computer Science and Engineering, University of Minnesota, MN, USA*
[1]{hendawi, Mokbel}@cs.umn.edu

[2]*Center for Data Science, Institute of Technology, University of Washington, WA, USA*
[2]{hendawi, mhali}@uw.edu

***Abstract*—This demo presents the *Panda* system for efficient support of a wide variety of *predictive* spatio-temporal queries. These queries are widely used in several applications including traffic management, location-based advertising, and store finders. *Panda* targets long-term query prediction as it relies on adapting a long-term prediction function to: (a) scale up to large number of moving objects, and (b) support predictive queries. *Panda* does not only aim to predict the query answer, but, it also aims to predict the incoming queries such that parts of the query answer can be precomputed before the query arrival. *Panda* maintains a tunable threshold that achieves a trade-off between the predictive query response time and the system overhead in precomputing the query answer. Equipped with a Graphical User Interface (GUI), audience can explore the *Panda* demo through issuing predictive queries over a moving set of objects on a map. In addition, they are able to follow the execution of such queries through an eye on the *Panda* execution engine.**

I. Introduction

The emergence of wireless communication networks and cell phone technologies with embedded global positioning systems (GPS) have resulted in a wide deployment of location-based services [7], [11]. Common examples of such services include range queries [3], [12], e.g., "find all gas stations within three miles of my *current* location" and K-nearest-neighbor (kNN) queries, e.g., "find the two nearest restaurants to my *current* location". However, such common examples focus on the *current* locations of moving objects. Another valuable set of location-based services focus on *predictive* queries [6], [8], [9], in which the same previous queries are asked, yet, for a *future* time instance, e.g., "find all gas stations that will be within three mile of my *future* location after 30 minutes". *Predictive* queries are beneficial in a wide variety of applications that include traffic management, e.g., predict congested areas before it take place, location-based advertising, e.g., predict the customers who are expected to be nearby in the next hour, and service finders, e.g., predict my closest gas station over the next hour of my route.

In this demo, we present the *Panda* system, designed to provide efficient support for *predictive* spatio-temporal queries. *Panda* provides the necessary infrastructure to support a wide variety of *predictive* queries that include predictive range queries and predictive k-NN queries for stationary and moving objects. *Panda* distinguishes itself from all previous attempts for processing predictive queries [9], [13] in the following: (1) *Panda* targets *long-term* predication in the order of tens of minutes, while existing attempts mainly target short-term prediction in terms of only minutes and seconds, (2) *Panda* does not only predict the query answer, but it also predicts the incoming queries, and prepares the result for these incoming queries beforehand, and (3) *Panda* is generic in the sense that it does not only address a certain type of predictive queries as done by previous work, instead, it provides a generic infrastructure for a wide variety of predictive queries.

The main idea of *Panda* is to monitor those space areas that are highly accessed using predictive queries. For such areas, *Panda* precomputes the prediction of objects being in these areas beforehand. Whenever a predictive query is received by *Panda*, *Panda* checks if parts of this predictive query are included in those precomputed space areas. If this is the case, *Panda* retrieves parts of its answer from the precomputed areas with a very low response time. For other parts of the incoming predictive query that are not included in the precomputed areas, *Panda* has to dispatch the full prediction module to find out the answer, which will take more time to compute. It is important to note here that *Panda* does not aim to predict the whole query answer, instead, *Panda* predicts the answer for certain areas of the space. Then, the overlap between the incoming query and the precomputed areas controls how efficient the query would be. This isolation between the precomputed area and the query area presents the main reason behind the generic nature of *Panda* as any type of predictive queries (e.g., range and kNN) can use the same precomputed areas to serve its own purpose. Another main reason for the isolation between the precomputed areas and queries is to provide a form of *shared execution* environment among various queries. If *Panda* would go for precomputing the answer of incoming queries, there would be significant redundant computations among overlapped query areas.

Panda provides a tunable threshold that provides a trade-off between the predictive query response time and the overhead for precomputing the answer of selected areas. At one extreme,

*This work is partially supported by the National Science Foundation, USA, under Grants IIS-0952977 and IIS-1218168.

we may precompute the query answer for all possible areas, which will provide a minimal response time, yet, a significant system overhead will be consumed for the precomputation and materialization of the answer. On the other extreme, we may not precompute any answer, which will provide a minimum system overhead, yet, an incoming predictive query will suffer the most due to the need of computing the query answer from scratch without any precomputations. Answer precomputation is similar to the idea of *pull* and *push* in publish/subscribe systems, however the distinction in *Panda* is that such service is offered in isolation from the incoming query and is also automated in a way that *Panda* smartly decides which areas to deploy *push* or *pull* methodologies.

The underlying prediction function deployed by *Panda* mainly relies on a *long-term* prediction function, designed by Microsoft Researchers to predict the final destination of a single user based on his/her current trajectory [2], [10]. Unfortunately, a direct deployment of such *long-term* prediction function does not scale up for a large number of moving objects nor it serves our purpose for predictive queries that are concerned with the moving object location in a future time rather than its final destination. *Panda* adapts such well-designed prediction function to: (a) scale up with the large number of users through a specially designed data structure shared by all moving objects, and (b) provide the prediction for a future query time (e.g., after 30 minutes) rather than only the prediction for the final destination. It is important to note here that a main goal behind *Panda* is to support long-term predictive queries as most existing work have only focused on short-term prediction.

The rest of this demo is organized as follows. Section II gives an overview of the *Panda* system including the system architecture, underlying data structure, and prediction function. Section III presents the main events that control the flow of execution in *Panda*, namely, object movement, query arrival, and statistics maintenance trigger. Finally, Section IV presents the demo scenario.

II. PANDA: SYSTEM OVERVIEW

This section gives an overview of the *Panda* system outlining the system architecture, the adaptation of the long-term prediction function [2], [10], and the underlying data structure. For detailed description of the *Panda* system, we refer the reader to [4], [5].

A. System Architecture

Figure 1 gives the system architecture of the *Panda* system, which includes three main modules, namely, answer maintenance, statistics maintenance, and query processing. Each module is dispatched by an event, namely, an object movement, a trigger for statistic maintenance, and a query arrival, respectively. *Panda* maintains a storage for precomputed answers, which is updated offline and used to construct the final query answer for arriving queries. Below is a brief overview of the actions taken by *Panda* for each event. Details of these actions are discussed in Section III.

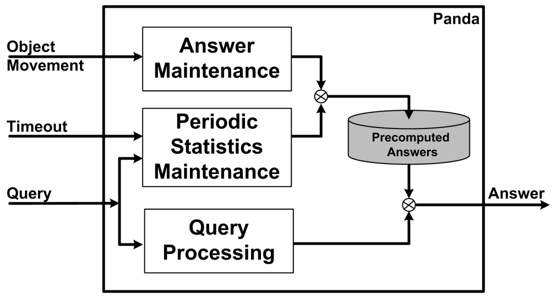


Fig. 1. The *Panda* System Architecture

Object movement. Whenever *Panda* receives an object movement, it dispatches the answer maintenance module to check if this movement affects any of the precomputed answers. If this is the case, the affected precomputed answers are updated accordingly.
Maintenance trigger. Based on a tunable threshold, a trigger may be fired to alert *Panda* that the current set of statistics that judge on which answers to precompute need to be reset. The updated statistics affect which parts of query answers will be precomputed.
Query arrival. Once a query is received by *Panda*, the query processor divides the query area into two parts based on the answer precomputation. The first part is already precomputed where its answer is just retrieved from the precomputed storage. The second part is not precomputed and needs to be evaluated from scratch through the computation of the prediction function against a candidate set of moving objects.

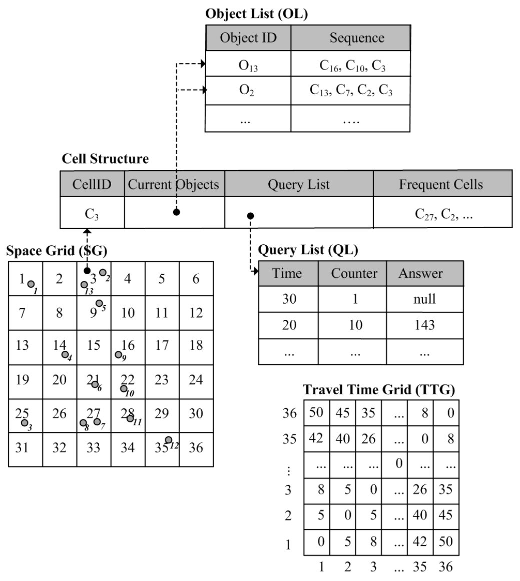
B. Prediction Function

The long-term prediction function deployed in *Panda* is mainly an adaptation of the one introduced by Microsoft Researchers to predict the final destination of a single object, F= $P(C_i|S_o)$ [2], [10]. F is applied to any space that is partitioned into a set of grid cells $\mathcal{C}$. F takes two inputs, namely, a cell $C_i \in \mathcal{C}$ and a sequence of cells $S_o = \{C_1, C_2, \cdots, C_k\}$ that represents the current trip of an object O. Then, F returns the probability that C_i will be the final destination of O.

As F only predicts the destination of an object, it does not have the sense of time. In other words, F cannot predict where an object will be after time period t. Since this is a core requirement in *Panda*, we adapt F to be able to compute the probability that object O will be passing by the given cell C_i after time t, where t is specified in the predictive query. The adaptation results in the function $\hat{F}$ which is a normalization of the results from the original prediction function F using the set of cells D_t that could be a possible destination of an object O after time t.

$$\hat{F} = \frac{P(C_i|S_o)}{\sum_{d \in D_t} P(C_d|S_o)} \tag{1}$$

Here, the numerator is the output of the original prediction function F, and the denominator is the summations of the probabilities of all grid cells in D_t, also computed from F. D_t is the set of possible destinations of object O after time t.



Object ID	Sequence
O_{13}	C_{16}, C_{10}, C_3
O_2	C_{13}, C_7, C_2, C_3
...	....

CellID	Current Objects	Query List	Frequent Cells
C_3	●	●	$C_{27}, C_2, ...$

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Time	Counter	Answer
30	1	null
20	10	143
...	...	...

36	50	45	35	...	8	0
35	42	40	26	...	0	8
⋮	...	...	...	0	...	...
3	8	5	0	...	26	35
2	5	0	5	...	40	45
1	0	5	8	...	42	50
	1	2	3	...	35	36

Fig. 2. Data Structures in *Panda*

Panda also has another adaptation of F to scale it up to support large numbers of moving objects as F is mainly designed to support single object prediction. The scaling up is mainly supported by the underlying data structure discussed in the next section which gives an infrastructure to share by large numbers of moving objects.

C. Data Structure

Figure 2 depicts the underlying data structure used by *Panda*. A brief overview of each data structure is outlined below:

Space Gird SG. *Panda* partitions the whole space into $N \times N$ grid cells. For each cell $C_i \in SG$, we maintain: (1) *CellID* as an identifier, (2) *Current Objects* as the list of moving objects located inside C_i, (3) *Query List* as the list of predictive queries issued on C_i. Each query Q in this list is presented by the tuple ($Time$, $Counter$, $Answer$, $Frequent$), where $Time$ is the future time included in Q, $Counter$ is the number of times that Q is issued, $Answer$ is the precomputed answer for Q which may have different format based on the type of Q, $Frequent$ is a boolean value indicating whether Q is precomputed or not, (4) *Frequent Cells* as the list of cells that one of their precomputed answers should be updated with the movement of an object in C_i.

Object List OL. This is a list of all moving objects in the system. For each object $O \in OL$, we keep track of an object identifier and the sequence of cells traversed by O in its current trip.

Travel Time Grid TTG. This is a two-dimensional array of $N^2 \times N^2$ cells where each cell $TTG[i, j]$ has the average travel time between space cells C_i and C_j, where C_i and C_j $\in SG$. TTG is fully pre-loaded into *Panda* and is a read-only data structure.

III. Event Processing in Panda

As depicted in Figure 1, *Panda* reacts to three main events, namely, object movement, query arrival, and a trigger for statistics maintenance. This section covers the actions of *Panda* for each event.

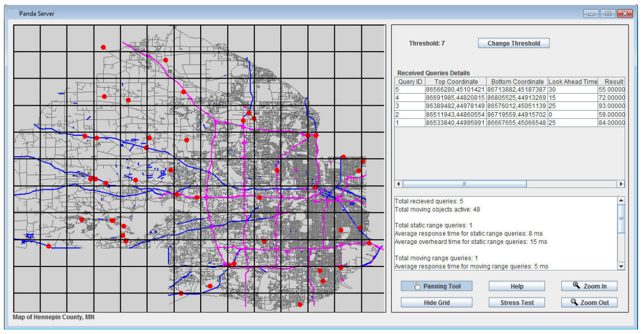
A. Object Movement

Once object O in cell C_i reports a change of location, the *answer maintenance* module is triggered to check on the new location of O. If the new location is still within cell C_i, then no actions will be taken by the *answer maintenance* module. However, if the new location of O is located in another cell C_j, the *answer maintenance* module checks to see if the movement of O affects any of the precomputed answer sets, and update the affected precomputed answers accordingly. This is mainly done as follows: For each cell c_f in the list of frequent cells of C_i, we calculate the prediction function $\hat{F}$ of O being in c_f for the times of interest of c_f, which can be obtained from the query list entry of c_f. For each time value, we update the value of the current answer by the value of the prediction function. For example, in the case of aggregate count queries, we subtract the result of the prediction function from the currently computed answer. We do the same for cell C_j, the new cell of object O, except for the fact that we add (instead of subtract) the value of the predication function to the current query answer.

B. Query Arrival

Once a new predictive query Q_t, asking about objects in a future time t, arrives to *Panda*, the *query processor* module is triggered to provide the answer in a very low response time. The *query processor* module first checks on the grid cells affected by Q_t. For the case of a range query, these cells will be the ones that overlap the query region. For a kNN query, these cells can be obtained by a pruning object that limits the set of cells to focus on.

For each affected cell c, we check its query list c_{ql} that includes the list of predictive queries issued on this cell. If the time t associated with Q_t is registered in c_{ql} with the *frequent* field set to *true*, then the part of Q that overlaps c is already precomputed, and we just retrieve its answer from the list c_{ql}. On the other side, if t is not registered in c_{ql} or if it is registered there, but with a false value in the *frequent* field, this means that the answer of this part of the query is not precomputed. In this case, we use the preloaded knowledge we have in the *Travel Time Grid* data structure along with the future query time t to apply a *time filter* over all grid cells in the space grid SG to find out those cells that are reachable to c within time t. For each moving object O in these filtered cells, we apply our prediction function $\hat{F}$ to calculate the probability that O will contribute to Q_t answer, and we update the query answer accordingly. In a nutshell, the answer for Q_t is collected from two parts: (1) A precomputed part based on the precomputed cells, and (2) The computation of the prediction function for moving objects that lie within grid cells reachable to the area of Q_t within time t.



Threshold: 7

Received Queries Details

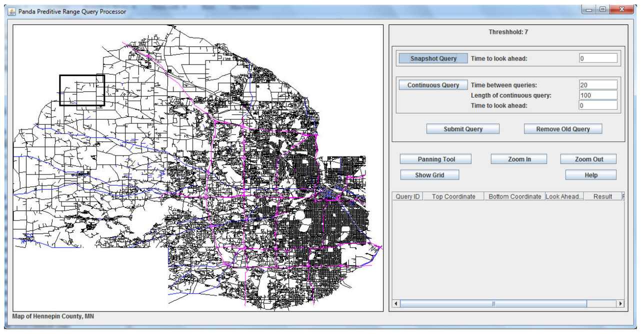
Query ID	Top Coordinate	Bottom Coordinate	Look Ahead Time	Result
5	86566280,45101421	86713882,45187387	30	55.00000
4	86691985,44820815	86805525,44913269	15	72.00000
3	86389482,44978149	86576012,45051139	25	93.00000
2	86511943,44860554	86719559,44915702	0	59.00000
1	86533840,44995991	86667655,45066548	25	84.00000

```
Total recieved queries: 5
Total moving objects active: 48

Total static range queries: 1
Average response time for static range queries: 8 ms
Average overheard time for static range queries: 15 ms

Total moving range queries: 1
Average response time for moving range queries: 5 ms
```

Fig. 3. System Internal Visualizer For *Panda*



Threshhold: 7

Time to look ahead: 0

Time between queries: 20
Length of continuous query: 100
Time to look ahead: 0

Query ID	Top Coordinate	Bottom Coordinate	Look Ahead...	Result

Fig. 4. Client User Interface

C. Statistics Maintenance Trigger

Two possible actions can trigger statistics maintenance:
Timeout. *Panda* needs to reset its statistics regularly. Every time period T_{out}, the counters of all queries are set to 0. This is mainly to make sure that the current counters present a recent image of the system in terms of what are the frequent queries.
Threshold change. At any point of time, a system administrator may change the threshold value $\mathcal{T}$. In this case, the statistics maintenance module is triggered to go through all query lists for all cells and update their *frequent* field for each time instance t according to the new threshold. For any update, the updated cell needs to be either populated or deleted for all reachable cells within time t. An updated cell c will be populated for time t if its *frequent* field got updated from *false* to *true*, or it will be deleted otherwise.

IV. Demo Scenario

This section presents the demo scenario of *Panda* that outlines the use of the *Panda* client and an eye on the internal operations of *Panda*. The demo is based on a large set of synthetic data of moving objects generated using the Brinkhoff's generator [1] on a real road network map extracted from the shape files of Hennepin County, Minnesota, USA. *Panda* has two main interface screens, each will be displayed on its own laptop during the demo session:
Client interface. Figure 4 gives a screen shot of the *Panda* client GUI. *Panda* users can use this interface to issue predictive spatio-temporal range and kNN queries by drawing the query area or location on the map, and entering the future time duration in the designated box. The answer is displayed back on the GUI along with the time taken by *Panda* to retrieve such answer. *Panda* users can ask to make their queries *continuous*, in which the answer of the query will be continuously updated to the user.
Server interface. Figure 3 gives a screen shot of the backend server of *Panda*. This is mainly done to help the demo audience understand the internals of the *Panda* system. It acts like an eye on the internal processing of *Panda*. The map area in this screen shot is partitioned into grid cells corresponding to the underlying data structure. Then, audience can monitor all moving objects on the map along with the received queries as they come. The answer for each query will be displayed on the right side of the interface along with a set of statistics showing the system behavior. The precomputed areas will be marked and illustrated. Audience can tune the system threshold $\mathcal{T}$ and see its effect on the system behavior. Finally, a preloaded query workload can be uploaded to stress test the system.

References

[1] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[2] J. Froehlich and J. Krumm. Route Prediction from Trip Observations. In *Society of Automotive Engineers (SAE) World Congress*, Michigan, USA, Apr. 2008.

[3] Y. Gu, G. Yu, N. Guo, and Y. Chen. Probabilistic Moving Range Query over RFID Spatio-temporal Data Streams. In *CIKM*, pages 1413–1416, Hong Kong, China, Nov. 2009.

[4] A. Hendawi. Predictive query processing on moving objects. In *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on Data Engineering ICDE*, pages 340–344, Illinois, USA, Mar. 2014.

[5] A. M. Hendawi and M. F. Mokbel. Panda: A Predictive Spatio-Temporal Query Processor. In *ACM SIGSPATIAL GIS*, California, USA, Nov. 2012.

[6] A. M. Hendawi and M. F. Mokbel. Predictive Spatio-Temporal Queries: A Comprehensive Survey and Future Directions. In *ACM SIGSPATIAL MobiGIS*, California, USA, Nov. 2012.

[7] H. Hu, J. Xu, and D. L. Lee. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In *SIGMOD*, pages 479–490, Maryland, USA, June 2005.

[8] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A Hybrid Prediction Model for Moving Objects. In *ICDE*, pages 70–79, Cancn, Mxico, Apr. 2008.

[9] H. Jeung, M. L. Yiu, X. Zhou, and C. S. Jensen. Path Prediction and Predictive Range Querying in Road Network Databases. *VLDB Journal*, 19(4):585–602, Aug. 2010.

[10] J. Krumm. Real Time Destination Prediction Based on Efficient Routes. In *SAE*, Michigan, USA, Apr. 2006.

[11] M. F. Mokbel, X. Xiong, M. A. Hammad, and W. G. Aref. Continuous Query Processing of Spatio-temporal Data Streams in PLACE. *GeoInformatica*, 9(4):343–365, Dec. 2005.

[12] H. Wang, R. Zimmermann, and W.-S. Ku. Distributed Continuous Range Query Processing on Moving Objects. In *DEXA*, pages 655–665, Krakow, Poland, Sept. 2006.

[13] R. Zhang, H. V. Jagadish, B. T. Dai, and K. Ramamohanarao. Optimized Algorithms for Predictive Range and KNN Queries on Moving Objects. *Information Systems*, 35(8):911–932, Dec. 2010.