Research Article

Location based Continuous Query Processing over Geo-streaming Data

K. V. Metre[1], M. U. Kharat[2]

[1]MET's Institute of Engineering, Nashik, India
[2]MET's Institute of Engineering, Nashik, India

Article History: Received: 10 November 2020; Revised: 12 January 2021; Accepted: 27 January 2021; Published online: 05 April 2021

Abstract: In recent years, many data-intensive and location based applications have emerged that need to process stream data in applications such as network monitoring, telecommunications data management, and sensor networks. Unlike regular queries, a continuous query exists for certain period of time and need to be continuously processed during this time. The algorithms used for data processing for the traditional database systems are not suited to tackle complex and various continuous queries over dynamic streaming data. The indexing for finite queries is preferred to indexing on infinite data to avoid expensive operations of index maintenance. Previous related work focused on moving queries on static objects or static queries on moving object. But now-a-days queries as well as objects are dynamic. So, hybrid indexing for queries significantly reduces the space costs and scales well with the increasing data. To deal with the speed of unbounded data, it is necessary to use data parallelism in query processing. The data parallelism in query processing offers better performance, availability and scalability.

Keywords: Data parallelism, hash value, space filling curve, spatial-temporal query

1. Introduction

In recent years, new data-intensive applications, such as network monitoring, traffic monitoring, sensor networks, telecommunications data management, and others, involve data streams. The processing of high volume data stream has cropped up as a research issue in various engineering and scientific applications e.g. monitoring mobile objects (Wang, 2011) tracking continuous network behaviors (Bohm, 2007) processing of data from sensors (Carney, 2002) and identification of RFID data (Park, 2007). To filter the data which is rapidly coming at the system as time-varying, unbounded sequences of data objects, the user uses continuous queries as against one-time submitted queries. These queries are processed over streams continuously for a period of time and provide new results incrementally on the arrival of new data tuples. The timestamps or time intervals are considered for system-wide windows for each single processing step (Golab, 2003). The authors introduced the adaptive windowing technique which dynamically resizes the window based on the incoming data (Bifet , 2007). (Babu, 2001) discussed requirements and challenges in query processing, and algorithmic issues.

The continuous queries are submitted once and executed continuously to get the output from data streams (Golab, 2003). One of the biggest challenges for a Data Stream Management System is to process continuous and massive data streams in consideration of the memory constraints for unbounded data. Window techniques focus on recent portion of the data. The recent data can be taken in terms of n objects or the objects during a time span. Re-execution of queries is a costly operation, and processing of continuous queries is considered an open issue in data stream systems using various indexing approaches. The data indexing becomes expensive as the data is not persistent, but it is in large volume. So, a new approach is suggested to build the index for queries which is finite rather than to build the index for data which is infinite (Carney, 2002). To evaluate location-based queries, the indexing for data or queries plays a key role. The various approaches are used for indexing spatial queries and spatial data which are categorized into (a) Tree-based structures (b) Grid structures and (c) Space-filing curves. Most applicable to the work presented here are space-filling curves and tree-based methods. The tree-based approaches such as R-tree (Prabhakar, 2002) incur a high updating cost due to tree structure. The grid-based methods use cell approach (Bohm, 2007; Kalashnikov, 2004) for indexing spatial queries and it divides the indexing pace into equal-sized grid. The grid-based approach results in better search and update time than the tree-based approaches (Kalashnikov, 2004) but it incurs high construction cost for indexing queries (Chen, 2011). Now days, due to dynamic nature of location-based queries, various query indexing approaches for continuous range queries (Gedik, 2006 ; Li & Karras, 2012) are suggested. Query Indexing mainly is suitable for evaluating continuous queries over dynamic and moving objects, as the queries exist for long periods of time, whereas the objects are continuously moving (Kalashnikov , 2002).

As the data is coming continuous and in unbounded manner, so it becomes a big challenge to filter massive data stream efficiently using query index. The query indexing method must support (1) scalability to massive data stream and (2) continuous queries. As we filter massive data which is coming continuously and the relevant objects for the given query are returned, it is rational to relinquish optimality with a good feasible solution that can be computed efficiently. The approximation methods provide reasonable and feasible solution instead of exact optimal solution.

Section 2 discusses previous work done in this area. Section 3 presents the hybrid indexing approach with related concepts. Section 4 discusses experimental setup and performance evaluation. Section 5 concludes the paper with conclusion.

2. Literature Review

The various approaches are used for indexing spatial queries and spatial data which are categorized into (a) Tree-based structures (b) Grid structures and (c) Space-filing curves. Most relevant to the work presented here are space-filling curves and tree-based techniques. In, tree-based concepts R-tree and its variants are used for indexing. The index on data objects as well as queries is built. It gives good performance only for small numbers of queries and unable to handle the arrival of new queries. More space is required due to both the type of indexing (Prabhakar, 2002). But, these methods have poor query performance due to dynamic nature of data and overlapping of data in tree nodes as it requires traversing the tree on more than one path to process the query. To index the continuous queries, some tree structures without overlapping are proposed (Park, 2007 ; Chen, 2011). The KDB-tree is used to index RFID queries (Park, 2007) which are continuous in nature. The concept of the KDB-tree is discussed in (Robinson, 1981) which provide single path traversing. The grid based methods use cell and cell structure to divide space of indexing into uniform cells for indexing queries (Kalashnikov, 2004). (Kalashnikov, 2004) used hierarchical grid cells to organize skewed queries to accelerate continuous range queries to process over dynamic spatial objects. But it suffers from high cost of storage and time for creation and updating of indexing structures (Chen, 2011).

A new index structure, called the Bkd-tree, a dynamic I/O-efficient data structure conceptually based on the technique used in kd-tree, is proposed (Vitter, 2003). Various authors suggested various indexing techniques such as R tree, R*-trees, motion-sensitive bounding boxes to index queries and data (Prabhakar, 2002). There is no need of regular updates to be made to the index structure due to query indexing. There is no need of any constraints on the speed of movement or path of the moving objects. Grid approach is used for query indexing. The authors experimented for uniform, skewed and hyper skewed queries and data using Grid index, R-tree Index (Kalashnikov, 2004). The index is used for less regularly changing queries and objects. The bounding boxes are updated only when queries and objects cross the predefined boundaries of their boxes. So, less updates to the index is required (Gedik, 2006). The authors proposed adaptive and cell-based data structure for indexing dynamic objects. It consumes about 30% of the storage space as against the storage space required by R-tree based methods. It achieves higher performance of query processing compared with the techniques based on R-tree (Chen, 2011). A dual-index structure with a pre computing approach for calculation of fast distance on changing objects. It incorporates a grid index fitting in memory and R-tree structure to save the road network information with their connectivity which is stationary to efficiently operate on the updates of the position of moving objects (Wang, 2011). A Distributed Strip Index, a data partitioning index and DKNN algorithm are used for distributed query processing. It results in integration overhead and the communication cost overhead (Ziqiang, 2015). The authors designed an efficient R-tree building algorithm giving better performance over sequential construction on GPU architecture(Luo & Wong, 2012). For indexing continuous range queries, GKDB (combination of grid and KDB tree) is proposed (Deng, 2015).

Using space filling curve n-dimensional space is mapped onto 1-dimension. The quality of a SFC is defined by its ability in conserving the locality of multi-dimensional objects mapped into linear order. Space filling curves include the Peano curve, Z-Order curve, Hilbert curve and many others. The Hilbert Curve is generally preferred for multi-dimensional region queries over the Z-Order Curve because of its superior clustering properties. Hilbert space-filling curve is selected to relate spatial space into the 1-dimensional domain, and multiple B+ trees are constructed (Aizawa, 2009). An algorithm for processing spatial range query algorithm is developed based on the combination of shifts and rotations using Hilbert curve (Jin, 2011). The scalability of the spatial queries using SFC keys improves while working with the data size compared to B-tree indexing. The queries using Hilbert values are faster than using Morton keys because of better spatial preservation nature of the Hilbert curve (Guan, 2018). An algorithm defined as quad-splitting is proposed to decompose a window on a Hilbert curve using various Hilbert levels avoiding the stages of merging and sorting. It resulted in lower time complexity compared to previous methods (Wu & Chang, 2008). The authors used two types of curves i.e. Hilbert and Peano to find the nearest neighbours. They provided transformation rules between Hilbert curve and Peano curve and it is observed that good clustering can be achieved (Chen & Chang, 2005). The sub-queries are generated from continuous queries using simulated annealing algorithm. The push and pull based data dissemination approaches are used to send the changes to the user as per user's interest (Thombare, 2014). The significant amount of temporal data is generated in network monitoring, stock exchange etc. which can be used in online decision making as dynamic data items are generated (M.Thombare, 2014). Various similarity

measures to find nearest objects are discussed (Kharat, 2018). Now days, Graphical processing Unit (GPU) plays a crucial role for processing data stream. It will accelerate the processing so that the task will be executed concurrently on host and device (Deshmukh, 2015). A CPU-based solution for continuous range-monitoring queries can be studied and the solution can be extended using the General Processing Unit (GPU). The input streams can be divided into parallel sub-streams, and queries are continuously executed on these sub-streams. The parallelization of continuous query execution with expensive operations can be achieved using GPU architecture.

3. Material and Methods

A hybrid indexing structure using Hilbert space and KDB tree (HKDB Tree) for continuous and dynamic spatial temporal region queries is proposed. The HKDB-tree meets the need of faster query processing to match to the rate of arrival of data stream containing spatial temporal objects.

3.1. Problem Formulation:

Firstly, the notations of a spatial-temporal data stream are provided. Then, continuous spatial-temporal query over the spatial data stream are defined.

Definition 1: Spatial-temporal object is defined as O = (id, t, loc)) where t is a timestamp and loc is a location of the object.

Definition 2: Spatial-temporal data stream is defined as

$S = \{ O_i \mid I \in [1,+\infty] \wedge O_i.t \leq O_{i+1}.t \}$ is an unbounded data set of spatial objects in timestamp order.

Definition 3: Continuous spatial-temporal query is denoted as q = (id, t_1, t_2, r) where t_1 and t_2 are the timestamps during which the query exists and r is the region of the query.

A Spatial-temporal object O in S matches q if the following two conditions are satisfied:

$$o.loc \in q.r \text{ and } q.t_1 \leq o.t \leq q.t_2 \quad (1)$$

For given a set Q of continuous spatial-temporal queries, for each incoming object O from a Spatial-temporal data stream S, a hybrid indexing using Hilbert curve and KDB tree technique over Q (HKDB) to rapidly deliver objects to spatial-temporal queries is proposed. Space Filling Curve such as Hilbert curve relates points from a multi-dimensional space to a 1- dimensional space.

3.2. Continuous Query Processing using H-KDB Tree (as shown in Figure 1)

Continuous spatial-temporal query processing consists of following phases:

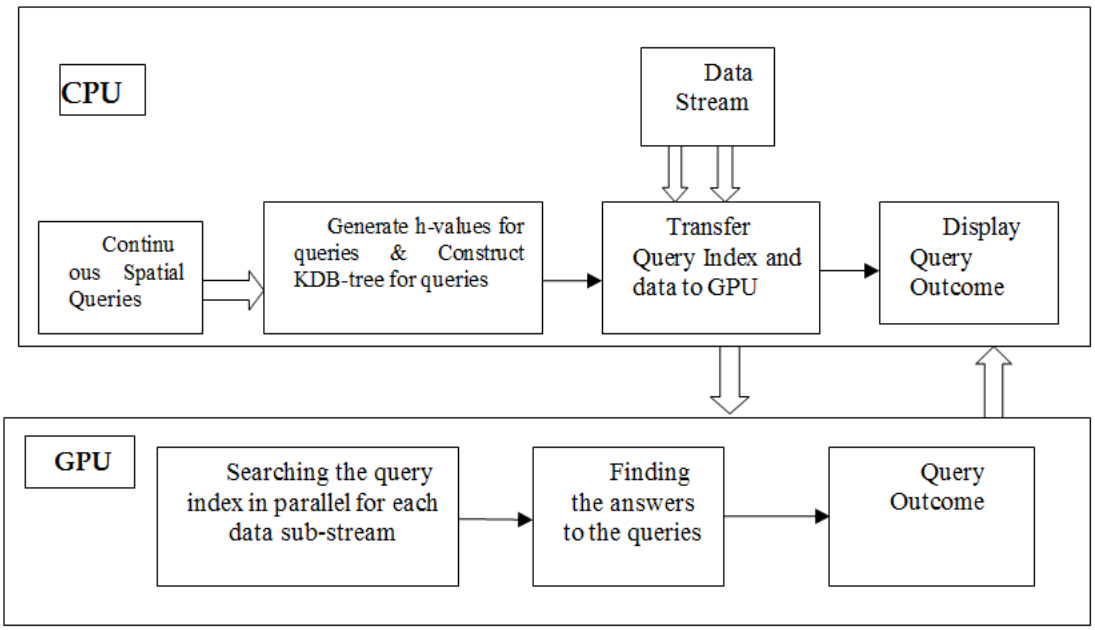

Figure 1. Working Model for Evaluation of Spatial-temporal Range Queries

1) Continuous Query Submission: The continuous queries are given by the user as per the definition stated above.

2) Acquisition of Data Stream: The infinite, unbounded datasets will be converted into bounded finite datasets using windowing technique

3) Construction of Index for Query/ Update: The hybrid indexing approach HKDB Tree is used for indexing the queries. The construction and updation of index will be performed by CPU.

4) Using data parallelism for execution of queries : Dividing the data stream into sub-streams to use the parallel processing capability of GPU to significantly speed up the execution.

5) Storing the result in a data stream: After the execution of the continuous query, the result of the query is stored in data stream.

The Hilbert curve is used to preserve the proximity of locality. It keeps the ordered and close points in adjacent locations in space. It is replicated in four quadrants. The 90 degrees clockwise rotation for lower left quadrant, 90 degrees anti-clockwise rotation for lower right quadrant is done. The traversal direction of left and right lower quadrants are reversed. But, no rotation and change in traversal direction for both upper quadrants. All these rotations and change in direction computation are dependent on the values obtained in the previous level.

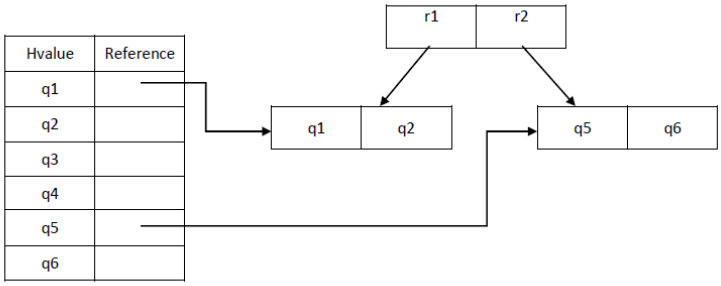

Figure 2. HKDB Tree indexing for queries

The below algorithm encodes x and y coordinates to a geohash of length precision on a corresponding a hilbert curve(h-value). Each character encodes bits per character (allowed are 2, 4 and 6 bits). Hence, the it encodes x & y using precision * bits_per_char. The number of bits devided by 2 give the level of the used hilbert curve, e.g. precision=10, bits_per_char=4, so total 40 bits and a level 20 of hilbert curve.

```
Algorithm  xy_to_hash(x, ,y, dim)
 //Output : it returns hash value on Hilbert curve
1.    begin
2.       d= 0
3.       level  = dim >> 1
4.       while (level > 0)
5.           rx = (x & level) > 0
6.           ry = y & level) > 0
7.           d += level * level * ((3 * rx) ^ ry)
8.           (x, y ) = rotate(level, x, y, rx, ry);
9.           level >>= 1;
10.       return  d;
11.   end

Algorithm  rotate(n,  x,  y,  rx,  ry)
1      begin
2      if ( ry==0)
3          if(rx==1)
4              x= n-1-x
5              y= n-1-y
6          else // swap x & y
7              t = x
8              x=y
9              y=t
10     return (  x, y )
11     end

Algorithm ObjectSearch  (Q : Query Dataset ,D: Set of Data objects )
//Output : A: Answer dataset
1      begin
2      Build KDB tree index for the range queries
3      Generate h-values for the range queries Q (set of Queries).
       Calculate the h-value for center of the range query
       Create an array H for index storing the h-value for each query in query dataset
         and reference to the query stored in KDB tree.
4      For every data object o ∈ D
          Calculate the h-value
          Select the query q from array H at minimum difference of h-value of the object
           and query.
                Map the object to the relevant range query such that
                    o.loc ∈ q.r and  q.t1 ≤  o.t ≤  q.t2
5       return A
6      end
```

3.3. Continuous Queries processing over Streaming Data with G-HKDB-Tree

Data stream is filtered using the index on continuous queries on GPU. The co-operation processing of both device and host is used. The CPU focuses on the task of creating and maintaining the index on queries while GPU focuses on processing data stream using G-HKDB tree in parallel. The different tasks are performed cooperatively 1) Inputting data stream into GPU memory 2) processing data stream using G-HKDB tree in parallel 3) Outputting query results back to CPU 4) maintaining query index on CPU.

For construction of KDB tree, no. of leaf nodes , N_{leaf} is computed as

$$N_{leaf} = (Max/m) \quad (2)$$

where Max is number of queries and m is capacity of leaf node.

The height of the tree h can be computed as

$$h = \log_M N_{leaf} \quad (3)$$

The streaming data is partitioned into different data segments. Let NS denote the count of GPU streams, Sp denote the size of each data partition/Segment.

The number of threads can be calculated as

$$|TH| = MAX\_TH / NS \quad (4)$$

The query index on host is built up using pointers and structure while the same index on GPU is built using sequential data structure i.e. arrays. The data in current window SCW is partitioned into S data segments. Let t_i, t_p and t_o denote the time required to input one data segment into GPU, to process this data using G-HKDBTree query index and to transfer the results back to CPU, respectively. The total time required to process continuous queries over SCW is defined as

$$T = S \times t_i + t_p + t_o \quad (5)$$

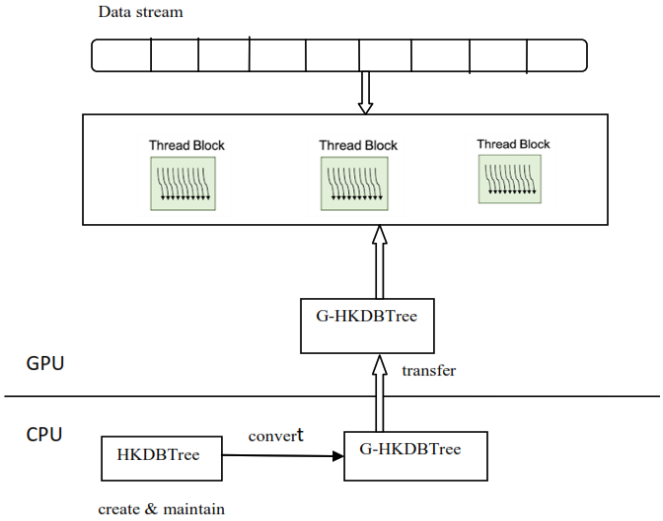

Figure 3. Host to device execution in cooperative way

```
Algorithm  Continuous Query Processing on data stream( G-HT, SCW)
//Input : G-HT : Query index on GPU, SCW : Data stream segments
//Output : AS : Answer set
1    begin
2        Partition SCW into P data partitions /segments seg[P]
3        P = (|SCW| / Sp)
4        Generate NS GPU streams GS[NS]
5        for (k=0; k<P; ++k) do
6           Input_Task_GPU( GS[k%NS], Seg[k])
7           Processing_Task_GPU (GS[k%NS], Seg[k], G-HT, AS)
8           Output_Task_GPU( GS[k%NS], AS)
9        end
10   return  AS
11   end

Processing_Task_GPU( GS, Sp,  G-HT, AS)
1  begin
     2   Assign GPU  threads TH for processing Sp
     3   for  each thread  tj  ∈  TH( 0 <= j <= |TH|-1)  do in  parallel
     4      for each data object  o ∈  Sp  do
5               Search_on_query_index(s, G-HT, AS)
6    end
```

4. Results and Discussion

The results of experimental evaluation are presented.

4.1. Experimental Setup

The geo-based services have been frequently used in various applications e. g. traffic monitoring, road-side assistance. To evaluate the proposed indexing approach, a synthetic data simulator is used to generate spatial moving objects in two- dimensional set using random waypoint model (Broch, 1998). The moving spatial objects are generated as points on unit square space [0, 1] X [0, 1]. The region size of the query is selected as 0.001. The experiments are run on i7-7500U @2.70 GHz with 8 GB RAM and GPU Platforms GeForce 940MX, Memory 2 GB DDR5, CUDA SDK 10.0.

4.2. Parameters of Experiments

Query dataset is of size 10000. For KDB Tree : 1) Capacity of an interior node M = 12, 2) Capacity of a leaf node m = 21.

To evaluate the performance of processing of the spatial queries on continuous data stream, mainly query execution cost is considered which plays a vital role for unbounded and massive data stream. The results i.e. execution time and index creation time are compared with the methods used in (Deng, 2015).

Table 1. . Query Performance

	Query Performance			
	Execution Time (Sec) on CPU		Execution Time (Sec.) on GPU	Speed up (x times)
Size of Dataset	GKDB Tree	HKDB Tree	HKDB Tree	HKDB Tree
100K	3.46	2.70	0.75	
200K	7.04	5.92	1.09	
300K	10.13	8.80	1.98	4.4
400K	14.02	12.09	2.77	
500K	18.14	16.09	3.61	

Table 2. Index creation Time

	Index creation Time (Sec)	
Size of Query Dataset	GKDB Tree	HKDB Tree
10K	6.3	5.3
20K	12.5	10.8
30K	19.8	18.9
40K	24.9	22.9
50K	29.4	28.0

To evaluate the performance of the spatial-temporal queries on dynamic data stream consisting of spatial-temporal data tuples, the two measures are considered i.e. query processing cost and index creation cost. The different alternative approaches of tree-based indexing data structures are used for indexing of data as well as queries. There is a trade-off between time complexity and space complexity. The data structures can support

different types of queries viz. circle query, range queries, or point queries. The range spatial- temporal queries are considered for the experimentation. As the hash values are calculated and stored for the queries and references t o the actual query in the KDB tree are maintained, the object searching becomes faster and it provides better query performance. In data stream-based applications, an approximation of solutions is preferred to exact solutions to cope with the rate of incoming objects in the system. In cell-based index structures, multiple entries of a query are stored if the query spans across multiple cells which results in higher index creation cost. In the discussed approach, the tree structure is used to store the queries avoiding the duplication of storing query in multiple cells and then in multiple trees. It results in less index creation cost. The methods provide reasonable and feasible approximate solution instead of exact optimal solution as the data is continuous and unbounded.

5. Conclusion

For efficient continuous query processing, it will be better to use the index for queries which is finite rather than to build the index for data which is infinite. This study explored query indexing approaches that can: (a) adapt to frequent changes of continuous queries and (b) achieve fast and scalable filtering of streaming data. To speed up the continuous query processing, the data parallelism approach is suggested where the data stream will be partitioned into different sub-streams and then the same queries will be executed in parallel on different sub-streams. For spatial- temporal queries, the proposed hybrid indexing approach shows 12% improvement in the query performance cost than the existing approach. There is an about 4.4x times speed up in query performance on GPU.

References

1. H. Wang & R. Zimmermann (2011). *Processing of continuous location-based range queries on moving objects in road networks*, IEEE Transaction on Knowledge and Data Engineering, vol. 23, no. 7, pp. 1065–1078,
2. C. Bohm, B. C. Ooi, C. Plant, & Y. Yan (2007). *Efficiently processing continuous k-NN queries on data Streams*, in Proceedings of IEEE 23rd International Conference on Data Engineering, pp. 156–165.
3. D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul & S. Zdonik (2002). *Monitoring streams-A new class of data management applications" in Proc. 28th Int. Conf. Very Large Data Bases*, pp. 215–226.
4. J. Park, B. Hong, & C. Ban. (2007). *A continuous query index for processing queries on RFID data stream*, in Proc. 13th IEEE International Conference on Embedded Real-Time Computation Syst. Appl., pp. 138–145.
5. L. Golab & M. T. € Ozsu. (2003). *Issues in data stream management*, ACM SIGMOD Rec., vol. 32**,** pp. 5–14.
6. Albert Bifet, Ricard Gavald. (2007). *Learning from Time-Changing Data with Adaptive Windowing*, Proceedings of the Seventh SIAM International Conference on Data Mining**.**
7. S. Babu & J. Widom.(2001). *Continuous queries over data streams*, ACM SIGMOD Rec., vol. 30, pp. 109–120.
8. S. Prabhakar, Y. Xia, D. Kalashnikov, W. G. Aref, & S. Hambrusch.(2002). *Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects*, IEEE Transactions on Comput., vol. 51, no. 10, pp. 1124–1140.
9. D. V. Kalashnikov, S. Prabhakar, W. G. Aref, & S. E. Hambrusch (2002). *Efficient evaluation of continuous range queries on moving objects*, in Proc. 13th Int. Conf. Database Expert Syst. Appl**.,** pp. 1–10.
10. D. V. Kalashnikov, S. Prabhakar, & S. E. Hambrusch (2004). *Main memory evaluation of monitoring queries over moving objects*, Distrib. Parallel Databases, vol. 15, pp. 117–132.
11. H. -L. Chen & Y. -I. Chang. (2011*). Nine-areas-tree-bit-patterns-based method for continuous range queries over moving objects*, IET Softw., vol. 5**,** pp. 54–69.
12. B. Gedik, K. -L. Wu, P. S. Yu, and L. Liu. (2006). *Processing moving queries over moving objects using motion-adaptive indexes*, IEEE Trans. Knowl. Data Eng., vol. 18, no. 5, pp. 651–668.
13. X. Li, P. Karras, L. Shi, K. -L. Tan, & C. S. Jensen (2012). *Cooperative scalable moving continuous query processing*, in Proc. IEEE 13th Int. Conf. Mobile Data Manage., pp. 69–78.
14. J.T. Robinson (1981). *The K-D-B-tree: A search structure for large multidimensional dynamic indexes*, in Proc. ACM SIGMOD Int. Conf. Manage. Data, pp. 10–18.
15. Jeffrey Scott Vitter, (2003). *Bkd-Tree: A Dynamic Scalable kd-Tree*, Springer-Verlag Berlin Heidelberg, pp 46-55.

16. Ziqiang Yu, Yang Liu, Xiaohui Yu, & Ken Q. Pu, (2015). *Scalable Distributed Processing of K Nearest Neighbor Queries over Moving Objects*, IEEE Transactions On Knowledge And Data Engineering, Vol. 27, No. 5, pp. 1383-1396.
17. L. Luo, M. D. Wong, & L. Leong, (2012). *Parallel implementation of Rtrees on the GPU*, in Proc. 17th Asia South Pacific Des. Autom. Conf., pp. 353–358.
18. Ze Deng, Xiaoming Wu, Lizhen Wang, Xiaodao Chen, Rajiv Ranjan, Albert Zomaya, & Dan Chen, (2015). *Parallel Processing of Dynamic Continuous Queries over Streaming Data Flows*, IEEE Transactions On Parallel And Distributed Systems, Vol. 26, No. 3, pp 834-846.
19. Kunio Aizawa & Shojiro Tanaka, (2009). *A constant-time algorithm for finding neighbours in quadtrees*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 31 No.7, pp. 1178–1183.
20. Ying Jin, Jing Dai & Chang-Tien Lu, (2011). *Efficient Range Query Using Multiple Hilbert Curves*, Current Trends and Challenges in RFID, Prof. Cornel Turcu (Ed.), ISBN: 978-953-307-356-9.
21. Xuefeng Guan, Peter van Oosterom, & Bo Cheng, (2018). *A Parallel N-Dimensional Space-Filling Curve Library and Its Application in Massive Point Cloud Management*, *ISPRS International Journal of Geo-Information*, 7, 327, p. 19.
22. C.-C. Wu Y.-I. Chang, (2008). *Quad-splitting algorithm for a window query on a Hilbert curve*, IET Image Processing, pp. 299-311.
23. Hue-Ling Chen, Ye-In Chang, (2005). *Neighbor-finding based on space-filling curves*, Information Systems 30 (Elsevier), pp. 205–226.
24. Manisha B. Thombare & K. V. Metre, (2014). *Query Optimization and Execution of Dynamic Data Items in Network Aggregation Environment*, Elsevier, pp. 1406 -1413.
25. M. Thombare & K. V. Metre, (2014). *Aggregation Environment for Query Optimization in Network Monitoring*, International Journal on Recent and Innovation Trends in Computing and Communication Volume: 2, Issue: 6**,** pp. 1515 – 1518.
26. M. U. Kharat, R. P. Dahake, & K. V. Metre, (2018). *Feature Dimension Reduction for Content-Based Image Identification*, IGI Global, Chapter**,** pp. 100-121.
27. K. B. Deshmukh & M. U. Kharat, (2015). *Accelerating Smith-Waterman Alignment Based on GPU*, Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 5, pp. 1162-1164.
28. J. Broch, D. A. Maltz, D. B. Johnson, Y. -C. Hu, & J. Jetcheva, (1998). *A performance comparison of multi-hop wireless ad hoc network routing protocols*, in Proc. 4th Annu. ACM/IEEE Int. Conf. Mobile Comput. Netw**.,** pp. 85–97.