International Journal of Engineering & Technology, 7 (4.19) (2018) 1060-1065

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Scalable Execution of KNN Queries using Data Parallelism Approach

Kalpana V. Metre[1]*, M. U. Kharat[2]

[1,2] *Department of Computer Engineering,*
MET's Institute of Engineering, Nashik, India.
**Corresponding author E-mail: kvmetre@gmail.com*

Abstract

In recent years, real-time data-oriented applications such as sensor networks, telecommunications data management, network monitoring are required to process various continuous queries on unbounded data streams. A lot of work has been done to deal with the computational complications in constant processing of continuous queries on unbounded, continuous data stream. The K-nearest neighbor algorithm (KNN) is a well-known learning method used in a wide range of problem-solving domains e.g., network monitoring, data mining, and image processing etc. The efficient and scalable processing of multiple continuous queries on dynamic data items requires query indexing and data indexing. Query processing algorithms used on static databases are not well suited to handle dynamic continuous queries over high dimensional data sets. It is better to build the index for queries which is finite rather than to build the index for data which is infinite. A divide-and-conquer approach is used for indexing and searching for K-nearest neighbors. The approach significantly will reduce the space complexity and will scale well with the increasing data size. The hybrid indexing approach using grid and a K-dimensional tree will reduce the space cost as well searching cost. The data parallelism will provide scalability of continuous queries over high-volume streams.

Keywords: *Data stream; Grid indexing; KNN; R-tree; Scalability*

1. Introduction

In recent years, continuous queries process real-time data streams in applications such as sensor networks, network monitoring, stock market etc. Unlike regular queries, a continuous query is evaluated regularly over a period of time. Traditional data processing algorithms are not well suited to handle various continuous queries over data streams. The previous researchers used either R-tree based indexing for data which is continuous and infinite. The maintenance and updating cost of those disk-based indexing is more than the query response time. Some of the earlier methods are not capable of handling the multi-dimensional data stream. If the response time of continuous query processing is larger than the rate of the input stream tuples, the delays are accumulated and it prevents the system from keeping up with the input stream rate. In many real-time applications, continuous KNN queries are processed on high dimensional dynamic data items. The K-nearest neighbor algorithm (KNN) is a well-known learning method or statistical search used in a numerous applications such as sensor networks, data mining and image processing etc. The advantages of KNN algorithm include that it is fairly simple to implement and it is well suited for multi-modal applications. The KNN search implementations have high computational costs, especially when used with a large amount of high dimensional data. Many KNN implementations degrade in performance as the data becomes high dimensional. Another drawback of KNN concerns its significant memory requirements, especially for indexing massive high dimensional data. The indexing techniques for the efficient and scalable processing of multiple continuous queries on dynamic data items include query indexing and data indexing. With dynamic data, frequent updates to the index on data are inevitable. 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. The rate of change of queries is low compared to that of data stream, so the query indexing methods can reduce the index maintenance cost [1]. This finite index on queries can be accommodated in memory which results in efficient execution of queries avoiding memory access frequently. The techniques used for building the index on queries include mostly grid indexing and tree-based indexing. The R-tree-based methods are suitable for location-based applications, but due to overlapping structure of tree, the methods discussed in [2] suffer from high maintenance cost. This approach gives good performance only for small numbers of queries.

The grid indexing divides [3][4][5] the d-dimensional indexing space into equal-sized cells and these cells are used for query indexing. It can give lower maintenance cost for dynamic queries and better performance than the other indexing such as tree-based methods [5]. The continuous queries can be static as well as dynamic. In applications such as location-based services, we need dynamic queries. The query indexing approaches need to tackle changing continuous queries [7] and KNN queries [10]. H. Wang et al. proposed an infrastructure MOVNET which combined R-tree structure to store the road network connectivity information and grid index to efficiently process moving object position updates. The limitation is that it can handle only stationary network [8]. W. Choi et al. introduced a new indexing structure that re-

fines cell structure adaptively with objects' movement to handle data skew and utilizes an overlapping technique to reduce the storage requirements [9]. Another challenge is to handle the volume and density of streaming data and filter massive high dimensional data using query index. To address these research challenges, the hybrid approach i.e. cell indexing and K-dimensional B-tree indexing for query indexing [6] for range queries is used so that the grid-based index will accommodate frequent updates properly and show a better performance than the tree-based approaches. The overlap-free tree approach requires less space so that the query index will be maintained in the main memory. The hybrid index structure will exploit the benefits of both the tree-based and the grid-based indexing structures.

The divide-and-conquer approach for indexing queries which are less compared to the massive dataset is proposed. Using the space partitioning approach, the queries will be indexed in grids. A focus of this work is to improve the performance of the KNN search and to demonstrate its performance in a real-world problem. It will support fast and exact KNN queries without scanning the entire data set for indexing. A solution based on a framework consisting of three major components is proposed: (1) Constructing the hybrid index for query points. (2) Searching for nearest neighbors for the continuous queries parallel, 3) Finding exact K-nearest neighbors for each and every query. Section 2 discusses related work done by various other researchers in this area. Section 3 presents in detail the combined approach and all the related concepts. Section 4 includes discussion and expected outcome. Section 5 concludes the paper with conclusion. The continuous query processing can be explained as shown in Figure 1.

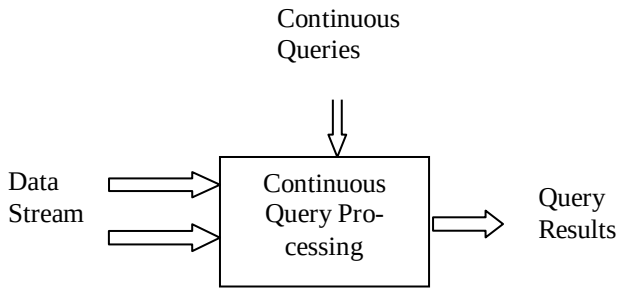
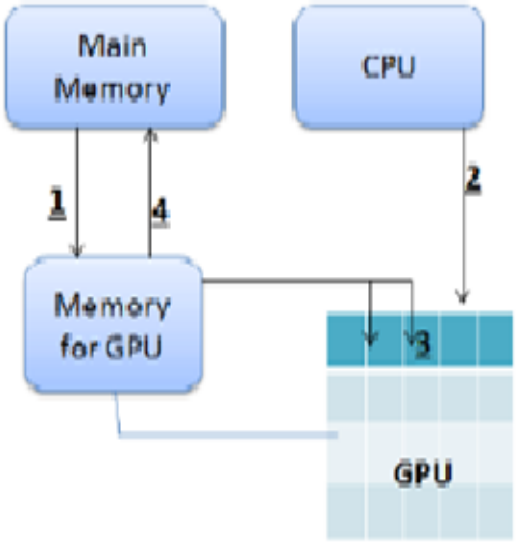


Fig. 1 : Continuous Query Processing

2. Related Work

The various research techniques have been proposed for finding KNN which has applications in various fields such as network monitoring, moving object recognition etc. The various researchers have proposed different indexing techniques for KNN search or similarity search. For KNN search, indexing techniques can be deployed either on massive high dimensional data or the queries submitted to the system. The data indexing techniques builds the index for high dimensional data streams. R-tree is used for indexing locations of moving objects for the execution of location based continuous queries [11]. The grid indexing method is used to divide the d-dimensional indexing data space into recursive cells to index position of moving objects [9]. Generally, the data indexing methods are preferred when the cost of index maintenance is insignificant [4]. But in data streams, data is continuous, frequent updates to the data index are necessary. The queries are finite and rate of change of queries is less than the continuous data streams. The query index can be accommodated in main memory whereas disk based indexing is required for the data. The indexing for queries can reduce storage cost as well as maintenance cost of index [5][1]. Therefore, the query indexing suits better for real-time applications. The query indexing methods are generally cell-based or tree-based. The tree-based approaches generally used the R-tree and its variants for indexing. The authors proposed R-tree based query indexing for monitoring moving objects [2]. However, it does not provide satisfactory query performance for data streams, because of the overlapping nature of non-leaf sibling of a R-tree. Every data object in a data-stream may have to traverse through the tree more than once for checking satisfying query conditions. To reduce the limitations of traditional tree-based indexing, a few non-overlapping tree structures were used for continuous queries' indexing [1]. Park et al. used KDB-tree indexing for RFID based continuous queries [1] and the KDB-tree provides overlap-free single-path traverse [12]. Due to the inherent structure of trees, the overlap-free tree requires high index maintenance cost. The cell indexing methods use grid and the indexing space is partitioned into equal sized grids/cells to index queries [3][4][5]. S. Prabhakar et al. used grid cells in hierarchical manner to organize skewed queries to speed up the execution of continuous range queries over moving objects [13]. In cell-based indexing, accessing the object requires O(1) query efficiency, cell-based indexing outperformed the tree-based approaches for query performance [5]. The authors used hierarchical cells to store queries which are skewed to speed up continuous range queries execution over moving objects. The costs of time and space are comparatively are high as the building and rebuilding of the indexing structures. The authors introduced a DSI –Distributed Strip Index (a data partitioning index) and DKNN algorithm for distributed query processing [14]. The clients process queries on the partial strip index provided by the server in parallel. It results in integration overhead and the communication cost overhead. The hybrid indexing is used for range queries which are used less frequently compared to KNN queries [6]. The index on data objects as well as queries is built. Re-evaluation of all queries with movement of objects is done keeping the constraint on velocity of object movement. It gives good performance only for small numbers of queries [2]. One of the limitations is that it cannot accommodate the arrival of new queries. The some of the previous methods are not capable of handling the multi-dimensional data stream. If the response time of continuous query processing is larger than the rate of the input stream tuples, the delays are accumulated and it prevents the system from keeping up with the input stream rate. To keep up with the rate of the input stream, an efficient data parallel and scalable system for executing KNN continuous queries can be used. A number of solutions have been introduced to reduce the cost of the KNN search. The quality and usefulness of these various solutions are limited by the computational time complexity of computing queries as well as the space complexity of the relevant search data structures. Data space partitioning can reduce redundant distance computations while searching for k nearest neighbours in a high dimensional data domain. The selection of appropriate data structures is an integral part of any efficient search algorithm design. The hybrid indexing technique balances both time and space complexities to achieve an overall reduction in both. One of the biggest challenges for a Data Stream Management System is to handle continuous data streams considering the memory constraints and without any random access to the 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 't' time span. L. Golab et al. used time interval or timestamps for generating windows for each single processing step [15]. As in [16], it is discussed that window is dynamically resized on the incoming data and a user-specifiable confidence value $\delta \in (0,1)$. A new method adaptive windows using sliding windows with re-computation of window size as per the rate of change in the incoming data is discussed. So, it is assumed that adaptive window stores the currently relevant data. The windowing can be done easily by using a fixed sliding window. Without disregarding the past data, recent data will always prominent influence on the computation to be performed. The batch-based partitioning strategy groups a number of consecutive windows into batch and assigns them to a particular partition. The key idea behind this strategy is to reduce the need for tuple replication to multiple partitions by reducing the overlap

across those partitions. It avoids repetition of tuples in a window [23].
Continuous queries are divided among the sub-queries. The simulated annealing algorithm was used for deriving sub-queries. The pull based and push based data dissemination mechanisms are used to send only those changes that are of interest to a user [17].
Using continuous queries, on-line decision making in applications such as network monitoring, stock exchange, often involves significant amount of time-varying data. In network monitoring data packet information is used as dynamic data items. This can be used for denial of service and intrusion detection [18]. The feature selection can be efficient and effective using clustering approach for finding the nearest neighbor from high dimensional dataset. Graph clustering method is used for feature selection [19]. In large multimedia database, for similarity search on images, the combination of textual pre-filtering and image re-ranking lists in a late fusion algorithm is used [20]. This approach provides the accuracy of the result. The authors have discussed various similarity measures to find the exact nearest neighbors. They discussed about clustering techniques for Content-Based Feature Extraction from Image [21]. Now days, Graphical processing Unit (GPU) receive lot of attention. Use of GPU is helpful to accelerate analysis task with a provision that the program will be executed concurrently on CPU and GPU [22]. The GPU works better for single instructions multiple data using data parallelism and it is suited for solving the problems in data stream management that can be expressed as data-parallel computations. By exploiting the parallel processing power of the GPU, it will be possible to significantly scale the number of dynamic data objects, while achieving an acceptable level of performance.



1) Copy processing data.
2) Instruct the processing.
3) Execute in parallel in each core.
4) Copy the result.

Figure 2: CPU-GPU processing flow [22]

3. System Overview

For continuous KNN query processing on unbounded data, query pre-processing and processing framework using a data-based parallelism approach is discussed. The graphical processing unit based design will achieve a high level of concurrency and it will also reduce the cost of communication and coordination. It will enable to handle very large datasets. The detailed system architecture is as shown in Figure 3.

This study focuses on the challenges of query processing in dynamic environment considering the dynamic continuous data as well as the changing queries.
Problem Formulation:
The data stream S consisting of continuous data set of tuples is

considered. S= {ti |i [1, + ∞)}.
Q is set queries which are dynamic.
MAX be the threshold value for the maximum number of queries.

Continuous KNN query processing consists of following phases:

1) Continuous KNN Query Submission:
The continuous queries of type K-Nearest Neighbors (KNN) are given by the user.

2) Acquisition of Data Stream and dividing data stream into sub-streams:

The dataset of dynamic data items or moving objects can be used. As the user is interested in recent and snapshot data of unbounded continuous data stream, it is necessary to use the windowing techniques. There are different techniques to extract recent data from the whole data set. In fixed sliding window technique, the windows can be either fixed by including only the most recent n data points or by taking only the most recent t time units of data (where n and t are constants). The selection of window size is very important.

3) Execution of KNN queries:

a) Construction of Index for Query/ Update of grid and KD tree :
The query preprocessing will be done to prepare a query index for all running queries. The hybrid approach is used for indexing i.e. grid based indexing and K-dimensional B tree indexing as shown in Figure 4. Whenever a new query is submitted or existing query is updated, the query index needs to be updated [6].

The queries are distributed into a number of grid cells. Let Q be a set of queries and a MAX be the threshold value for the maximum number of queries in a grid cell. The grid cell will be divided into sub cells in d-dimensional space used for indexing. The cell having the queries less than the threshold value MAX will not be partitioned into sub cells and a cell having queries greater than the threshold value MAX, will be divided into sub cells. The partitioning of cell into sub cells will be continued recursively until no cell has queries greater than threshold value MAX. The KDB tree index will be built for every grid cell where the each node will denote range boundaries for the particular cell. The KDB-tree consists of leaf nodes and internal nodes. Each internal node represents the region of the cell and contains maximum M index values. The index entries contain the region of child. The regions in all the nodes at the same level are not overlapping. Each leaf node contains actual query data i. e. m query entries in the particular cell.

b) Finding the exact KNN objects:

The filtering of the data object will be done in parallel on sub-streams. For each element from the data sub-stream, the cell will be identified and the KDB tree for that cell will used to search for the query for the data element and it will be done in parallel for each sub-stream. The distance between data element and query is calculated using following formula to find out exact KNN objects from data stream.

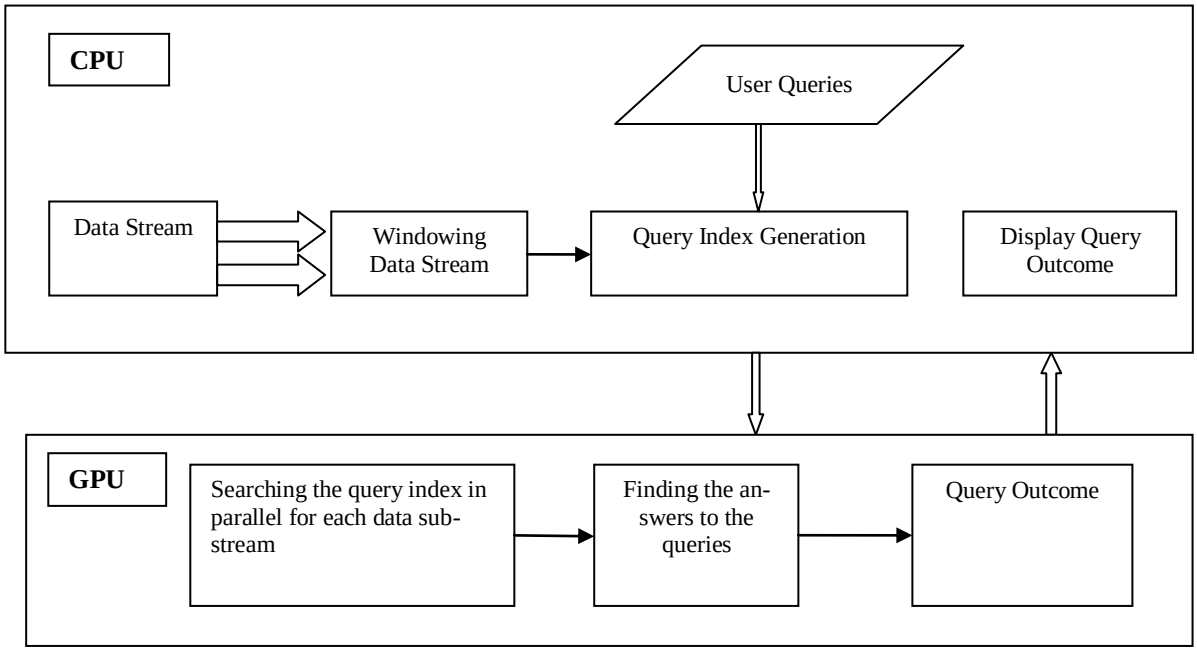


Fig. 3: Query indexing and Data-based parallelism for Continuous Query Processing

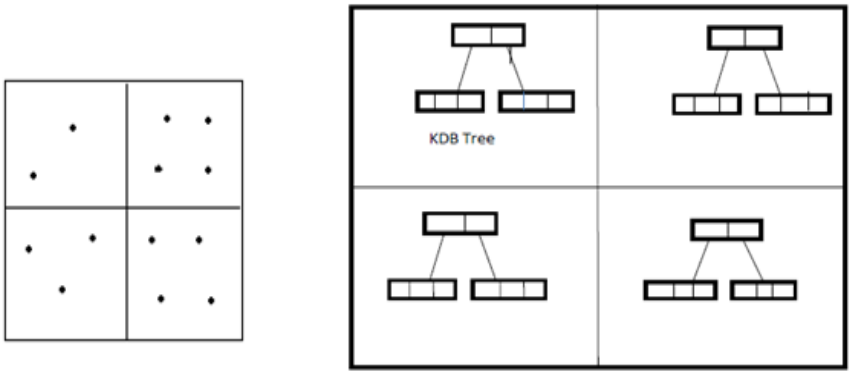


Fig. 4: a) Grid Indexing for Queries b) KDB Tree Indexing for each cell

Euclidean distance:

$$d(q,d) = \sqrt{\sum_{i=1}^{d} (q_i - d_i)^2}$$

The Euclidean distance is calculated between each data object in the set of objects for the query and the query object.

Algorithm for updating the query index :

```
Algorithm UpdateQueryIndex ( T, q, q1)
//Input: T - CKDB-Tree for Query Index, q - query to be changed,
q1 - new query
// Output: Updated T
begin
cell = locateCell( T, q)
for cell ∈ T
node = locateNode( cell.tree, q)
if (node != NULL) then
Delete q from cell.tree and insert q1 into
cell.tree
end
end
```

Algorithm for searching the query index:

```
Algorithm SearchQueryIndex ( T, DS)
// Input: T - CKDB-Tree for Query Index, DS – Data sub-stream
// Output: ResultSet
begin
for each data object t ∈ DS
cell = locateCell( T, t )
searchKDB-tree (cell.tree.root, t, ResultSet,d )
end
return ResultSet
end
Algorithm searchKDB-tree (node, t, ResultSet , d )
//Input : t – node of KDB tree, d- threshold value for Euclidean
distance
// Output : Resultset
begin
 if ( node is a leaf node) then
for (each query data q in node) do
if ( dist ( q, t ) < d) then
ResultSet = Union ( ResultSet, t )
end
 else
```

node = findChildNode(t, node.children)
searchKDB-tree (node, t, ResultSet,d)
end

The update cost (upper bound) for query is computed from the height of the tree h. The number of leaves in KDB-tree N_{leaf} is computed as

$$N_{leaf} = (Max /m) \quad (1)$$

where Max is threshold value for the maximum number of queries in a grid cell and m is capacity of leaf node.
The number of query indexes in (h-1) level is estimated as

$$N_{(h-1)} = N_{leaf} / M \quad (2)$$

where M is the capacity of internal node in the tree.
The height of the tree h can be computed as

$$h = [\log_M Nleaf] \quad (3)$$

4. Discussion and Expected Outcome

To evaluate the queries on continuous data stream, where the queries are also changing, the three measures are considered i.e. index storage cost, index maintenance/update cost and query execution cost. The various alternatives of R-tree and B-tree are used for the indexing of queries as well as data. Some index structures have less space complexity and others may have less time complexity and they support different types of queries viz. range queries , circle query or point queries and different data types of datasets such as linear or multidimensional data. Most of the tree based index support point query to find nearest neighbors and single dimensional data efficiently. But in case of processing of continuous queries on multidimensional data, specific data structure is required. B-tree and its variations support point query and single dimensional data efficiently while R-tree and its alternatives support multidimensional data and range query efficiently. In R-tree, due to overlapping of non-leaf nodes, traversing a tree for a query by a data tuple can be more than one to get the result. This technique results in high maintenance cost in case of dynamic queries. Using B+tree, it is possible to store large key values. The time required to search a particular key from the tree is proportional to the height of the B-tree and it will be comparatively less. But, it is not preferable to use B-tree for the queries which consist of searching on multiple keys. This technique also results in high maintenance cost in case of dynamic queries. The adding or removing queries is done very efficiently with the grid of cells. This indexing is proved to outperform the R-tree based query index. During query reevaluation, query lists are used to find all the queries. As the number of cells is increased, it reduces the average number of queries for a cell thereby reducing the processing time. The KDB-tree usually outperforms than other multi-dimensional indexes for a query, because it provides a single-path traverse of the tree. Dynamic insertion and deletion of object is balancing. The searching the query in KDB-tree can be done efficiently. KDB-trees combine properties of KD-trees and B-trees. The multidimensional search efficiency of balanced KD-trees and the I/O efficiency of B-trees is combined in the KDB-tree.
As the queries are not static, the index maintenance cost using the hybrid approach will be significantly less compared to tree based approaches where the frequent restructuring of the tree will take place with changing queries. The hybrid approach will show the relatively better performance in storage cost and query execution cost compared to cell indexing as the overlap-free K-dimensional tree structure is used

5. Conclusion

For continuous query processing, it is suggested to build the index for queries which is finite rather than to build the index for data which is infinite. The rate of change of queries is low compared to the data stream, so the query indexing methods can reduce index maintenance cost. The divide and conquer approach can be used for the index generation for queries. The hybrid indexing approach will outperform the existing techniques for continuous query processing for the data stream of dynamic data items. The parallel filtering of data objects will provide scalability of continuous KNN queries on massive high dimensional data with more accuracy. This technique of indexing and processing of queries balances both time and space complexities to achieve an overall reduction in both.

References

[1] J. Park, B. Hong, and C. Ban, “A continuous query index for processing queries on RFID data stream,” Proceeding. of 13th IEEE International Conf erence on Embedded Real-Time Comput. Syst. Appl., (2007), pp. 138–145.

[2] S. Prabhakar, Y. Xia, D. Kalashnikov, W. G. Aref, and S. Hambrusch, “Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects,” IEEE Transactions on Comuters, vol. 51, no. 10, (Oct. 2002), pp. 1124–1140.

[3] C. Bohm, B. C. Ooi, C. Plant, and Y. Yan, “Efficiently processing continuous k-NN queries on data streams,” in Proceeding of IEEE 23rd International Conference on Data Engineering., (2007), pp. 156– 165.

[4] X. Yu, K. Q. Pu, and N. Koudas, “Monitoring k-nearest neighbour queries over moving objects,” in Proceeding of . 21st International Conference on Data Engineering, (2005), pp. 631–642.

[5] D. V. Kalashnikov, S. Prabhakar, and S. E. Hambrusch, “Main memory evaluation of monitoring queries over moving objects,”Distributed Parallel Databases, vol. 15, (2004), pp. 117–132.

[6] Ze Deng, Xiaoming Wu, Lizhen Wang, Xiaodao Chen, Rajiv Ranjan, Albert Zomaya, and Dan Chen, “Parallel Processing of Dynamic Continuous Queries over Streaming Data Flows”, IEEE Transac tions On Parallel And Distributed Systems, Vol. 26, No. 3, (March 2015) .pp. 834-846.

[7] B. Gedik, K. -L. Wu, P. S. Yu, and L. Liu, “Processing moving queries over moving objects using motion-adaptive indexes,”IEEE Transaction on Knowledge and Data Engineering., vol. 18, no. 5, (June 2006), pp. 651–668.

[8] Haojun Wang and Roger Zimmermann, “ Processing of Continuous Location-Based Range Queries on Moving Objects in Road Networks “, IEEE Transactions On Knowledge And Data Engineering, Vol. 23, No. 7, (July 2011), pp. 1065-1078.

[9] W. Choi, B. Moon, and S. Lee, “Adaptive cell-based index for mov ing objects,” Data and Knowledge Engineering, vol. 48, (2004, pp. 75–101.

[10] F. Liu and K. A. Hua, “Moving query monitoring in spatial network environments,” Mobile Netw. Appl., vol. 17, pp. 234–254,2012.

[11] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez, “Indexing the positions of continuously moving objects,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, (2000), pp. 331–342.

[12] J.T. Robinson, “The K-D-B-tree: A search structure for large multidimensional dynamic indexes,” in Proc. ACM Sigmod International. Conference on Management of Data, (1981), pp. 10–18.

[13] D. V. Kalashnikov, S. Prabhakar, W. G. Aref, and S. E. Hambrusch, “Efficient evaluation of continuous range queries on movingobjects,” in Proc. 13th International Conference on Database Expert System Applications, (2002), pp. 1–10.

[14] Ziqiang Yu, Yang Liu, Xiaohui Yu, and Ken Q. Pu, “Scalable Distributed Processing of K Nearest Neighbor Queries over Moving Objects”, IEEE Transactions On Knowledge and DataEngineering, Vol.27, No.5, (May 2015), pp. 1383-1396.

[15] L. Golab and M. T. € Ozsu, “ Issues in Data stream management ” ACM SIGMOD Rec., vol. 32, (2003), pp. 5–14.

[16] Albert Bifet, Ricard Gavald, “Learning from Time-Changing Data with Adaptive Windowing” Proceedings of the Seventh SIAM International Conference on Data Mining, (2007)

[17] Manisha B. Thombare, K. V. Metre, “Query Optimization and Execution of Dynamic Data Items in Network Aggregation Environment”, Elsevier , (2014) , pp. 1406 -1413.

[18] M. Thombare, K. V. Metre, " Aggregation Environment for Query Optimization in Network Monitoring", International Journal on Recent and Innovation Trends in Computing and Communication Volume: 2,Issue: 6 (2014), pp. 1515 – 1518.
[19] H. D. Gangurde, K. V. Metre, "Clustering based Feature Selection from High Dimensional Data", International Journal on Recent and Innovation Trends in Computing and Communication , Volume: 3 , Issue: 6 , (2015), pp. 3556 – 3560.
[20] Trupti Atre, K. V. Metre, " MIRS: Text Based and Content Based Image Retrieval International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 3, Issue 4, (2014), pp. 579-584.
[21] M. U. Kharat, R. P. Dahake, K. V. Metre, Feature Dimension Reduction for Content-Based Image Identification, Book Chapter (Clustering Techniques for Content-Based Feature Extraction From Image) , IGI Global, (2018), pp. 100-121.
[22] K. B. Deshmukh, M. U. Kharat, "Accelerating Smith-Waterman Alignment Based on GPU", Journal of Advanced Research in Computer Science and Software Engineering", Volume 5, Issue 5, (2015), pp. 1162-1164.
[23] Cagri Balkesen and Nesime Tatbul, "Scalable Data Partitioning Techniques for Parallel Sliding Window Processing over Data Streams", 8th International Workshop on Data Management for Sensor Networks (DMSN) (2011)