CONTINUOUS QUERY PROCESSING IN A DATA STREAM ENVIRONMENT

Dong Gyu Lee*, Bong Jae Lee**, Keun Ho Ryu*

*Chungbuk National University, dglee@dblab.chungbuk.ac.kr
*Chungbuk National University, khryu@dblab.chungbuk.ac.kr
**Korea Electric Power Research Institute, bjlee@kepri.re.kr

ABSTRACT ... Many continuous queries are important to be process efficiently in a data stream environment. It is applied a query index technique that takes linear performance irrespective of the number and width of intervals for processing many continuous queries. Previous researches are not able to support the dynamic insertion and deletion to arrange intervals for constructing an index previously. It shows that the insertion and search performance is slowed by the number and width of interval inserted. Many intervals have to be inserted and searched linearly in a data stream environment. Therefore, we propose Hashed Multiple Lists in order to process continuous queries linearly. Proposed technique shows fast linear search performance. It can be utilized the systems applying a sensor network, and pre-processing technique of spatiotemporal data mining.

KEY WORDS: Continuous Query, Query Processing, Data stream, Concurrent query, Query index

1. INTRODUCTION

There are ongoing various researches for building an ad-hoc network applying sensors in real world to development of sensor network and mobile computing technologies[1, 2]. To provide intelligent ubiquitous services based on the network as this, once it needs to query and combine incoming data in sensor nodes. However, it is inadequate to process a data stream continuously occurring in sensor network by existing language and operations for processing static data.

There are two different between sensor data and existing database[1]. First, data stream is processed continuously. Events requiring a response immediately as Traffic accidents are processed in near real time. Because raw data stream for saving in disk requires expensive storage cost. Second, since sensor nodes have a low performance processor and limited power resource, query processor needs to reduce consumption of power resource. To monitor a data stream and process efficiently, a lot of continuous queries can be made and evaluated continuously against each data item in the incoming data stream[3].

Query index techniques are used to process continuous queries quickly. Query index storing the intervals in order to process queries efficiently checks intervals containing search value and then returns query identifiers. Query index of good performance must have low storage cost and fast linear search performance for processing continuous queries in a data stream environment. It must also store a number of queries. Figure 1 is a structure applying the query index. The query index, query analyzer, query response, and message DB are consisted in server side. If the users send queries to query analyzer, it checks a condition of query and makes an interval of each attribute.

A query index consists of created interval according to attribute in query analyzer and the query index returns query identifiers searching interval against incoming data stream in sensor network. Query response searches messages in message database using identifiers of each attributes returned. It prevents natural disasters operating actuator or inform to users through the messages. Such structure enables to apply data stream management system or context-awareness system and they have an advantage to process concurrently a number of queries by query index.

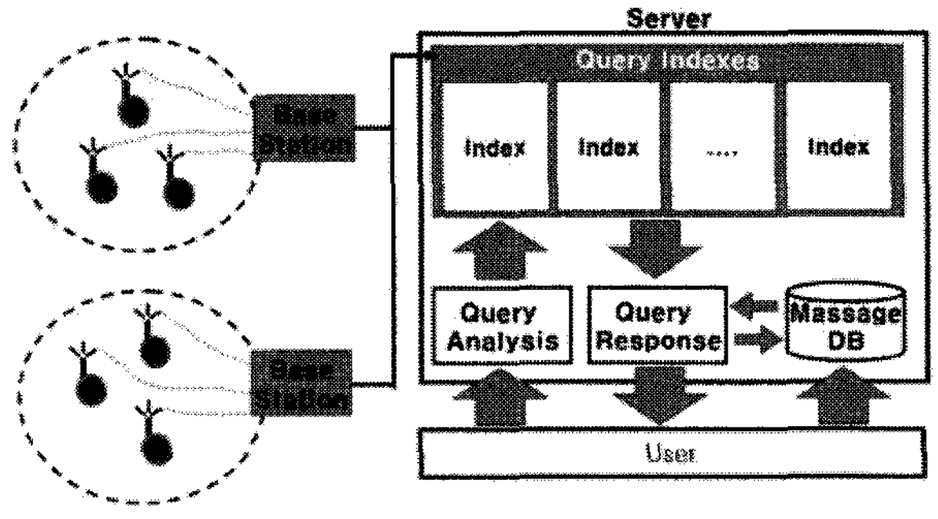

Figure 1 Architecture for multiple query processing

We can check to be processed continuous queries efficiently by query index using such structure for application in a data stream environment. If the structure does not have a query index, it must process continuous query one by one or it consumes large storage cost and search cost to process after storing continuous query in database. The query index needs for processing continuous query efficiently in a data stream environment. Therefore, we propose Hashed Multiple Lists for evaluating continuous query efficiently. Proposed technique has linear fast search and enables to apply it to

data stream management system or context-awareness system.

2. RELATED WORK

As the existing research of query index, the indexes performing stabbing query[3, 6] that finds interval containing a numerical value and CEI-based query indexing[3] having fast linear search performance in a data stream environment are explained in this section. There are Segment trees[6], Interval trees[6], Interval binary search trees(IBS-tree)[7], and Interval skip lists(IS-lists)[8] as existing researches.

Existing researches is not stable to be not considered in a data stream environment. Segment tree and interval tree is difficult to insert and delete dynamically that they already consist of all of interval. IBS-tree and IS-lists are designed a query index technique based on main-memory.

Two techniques are first dynamic approaches managing the large number of overlapped intervals. Though they are similar with principle, the implementation of IS-lists is simpler and its performance is better. They can express various intervals of queries and can be apply context information filter that transfers messages to the user according to rules. If the number of inserted intervals is large and search value is far from header, their search time is increased. When long interval is inserted, insertion cost is large. If n is the total number of IBS-tree and IS-lists, they require search time of $O(\log(n))$ and storage cost of $O(n\log(n))$.

Recently, there is CEI-based query index technique for processing continuous queries efficiently in a data stream environment. This technique has fast linear search performance, low storage cost, insertion and deletion dynamically differing with existing researches as stabbing query. However, if the total number of interval is less than 100 thousand and storage cost is larger than IS-lists, insertion cost of long interval is also larger. When long intervals are inserted in CEI-based query indexing, it has low search performance to insert same query identifiers much in identifier list of this index.

Therefore, search cost of existing researches is increased in proportion to the number and length of inserted intervals. It is difficult to process a lot of intervals with wide width.

3. PROPOSED ALGORITHMS

HMLists consists of insertion, deletion, search, and level adjustment algorithm. However, we explain each step of search and level adjustment algorithm in this paper.

3.1 Search Algorithm

Figure 2 is the search algorithm that finds query identifiers containing search value against data stream. If a node of level 4 contains a search value of figure 2, it returns query identifiers irrespective of next node and descends to lower step. If a value of node is less than search value, it descends the level and repeats the operation that returns query identifiers. If a node containing search value does not exist, it returns identifiers of current pointer. If existing, it returns query identifiers of the node. Search cost of this algorithm is $O(\log n)$.

```
Algorithm HML_Search(K)
INPUT   K: Search key
OUTPUT  S: set of searched query IDs
BEGIN
S=Ø
i=maximum level
h=K/8
x=8h node allocating hashtable[h]
 WHILE(i≥ 0 and (x is 4 level's node and x→key≠ K)) DO
  S=S∪x→edgeID[i]
  i=i--
  WHILE(x→forward[i]≠ null
  and x→forward[i]→key<K) DO
   x=x→forward[i]
  ENDWHILE
  IF(x is not the header and x→key≠ K) THEN
   S=S∪x→edgeID[i]
  ELSE IF(x is not the header) THEN
   S=S∪x→ID[i]
  i=i--
 ENDWHILE
END
```

Figure 2 Search algorithm

3.2 Level Adjustment Algorithm

Figure 3 shows that an interval has minimum value and maximum value of an attribute and each value is value of a node. Each node has the levels and the level of node is decided as structure of binary tree. Algorithm of figure 3 is algorithm that consisted of structure of binary tree that Height of HMLists is 4 and the number of leaf node is 8. It decides the level calculating the rest by Simple formula that adjusts from highest level to lowest level.

```
Algorithm HML_AdjustLevel(K)
INPUT   K: Key for adjusting level
OUTPUT  Level: Level of K
BEGIN
 n=0, Level=0
 H=height of HMLists
 L=K%2^(H-1)
 IF(L=0) THEN Level=H-1
 ELSE IF(L=H) THEN Level=H-2
 ELSE IF(L=H-2 || L= H+2) THEN Level=H-3
 ELSE Level=0
END
```

Figure 3 Level Adjustment algorithm

4. PERFORMANCE EVALUATION

This paper evaluates 3 techniques such as IS-lists, CEI-based query index, and Hashed Multiple Lists. The approaches are implemented by C++ language in

windows system of CPU 2.0GHz, and RAM 1GB memory.

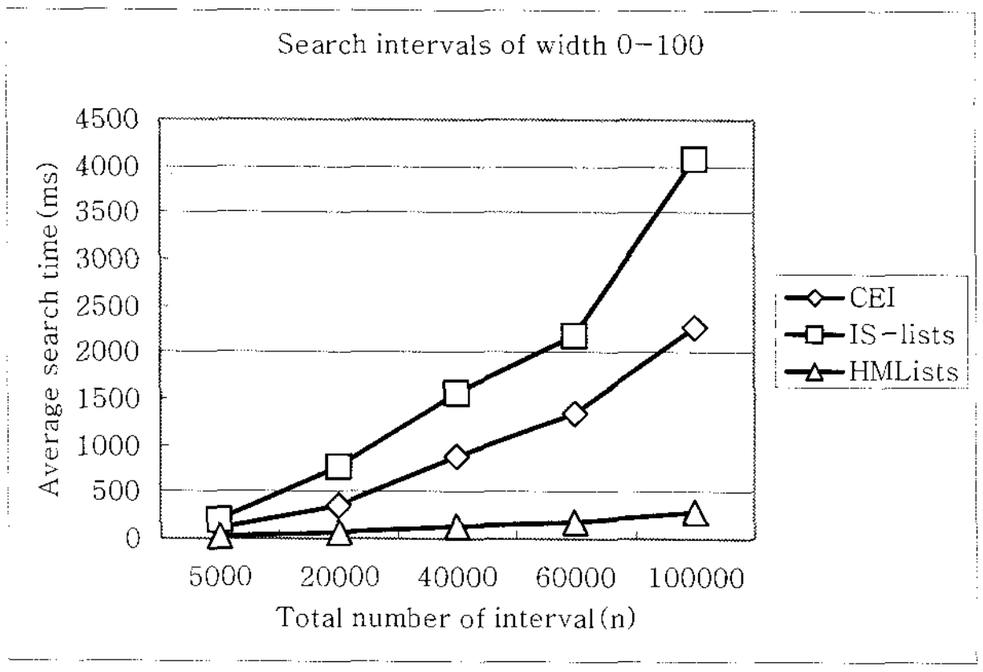

Figure 4 Average search time to width 0-100

Figure 4 set up the interval width of queries between minimum value 0 and maximum value 100. The indexes are consisted of intervals between 5 thousand and 100 thousand and then average search time of the indexes is evaluated. IS-Lists approach performs search operation from root node progressively. If search value is far from root node and the number of intervals is increased, it takes long time. Average search time of CEI-based query index is grown since the number of queries is increased. It shows that others comparing with proposed approach are not stable. Though search time of CEI-based query index takes long time increasing the number of queries, it is bigger problem that the index can not search linearly irrespective of a number of queries. However proposed technique reduces the number of pointer limiting to level of a node 4 and searches all search value within maximum 8 nodes linearly using hash table.

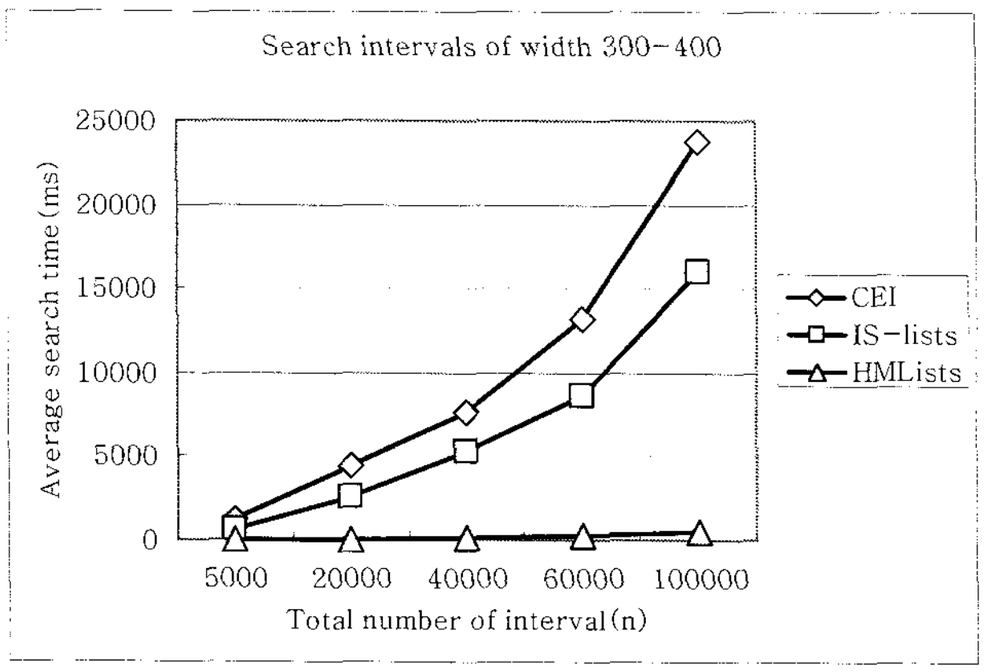

Figure 5 Average search time to width 300-400

Figure 5 shows CEI-based query index performs worst performance setting up the interval width of queries between minimum 300 and maximum 400. Though figure 4 set up the interval width of queries between 0 and 100, figure 5 set up the wider interval width of queries between 300 and 400. Since CEI-based query index search query identifiers on this condition, it requires much search time. Also, since IS-lists search from root node to NULL node, it takes much search time. However, proposed Hashed Multiple Lists can search within maximum 8 nodes linearly.

5. CONCLUSION

We proposed Hashed Multiple Lists for processing efficiently continuous query in a data stream environment. Proposed approach shows fast linear search performance minimizing the number of node accesses since it decides the level of node as the structure of binary tree and uses hash table. Although the number of queries is increased, it can search linearly through hash table. Proposed technique is applied data stream management system or context-awareness system. Ongoing work needs to evaluate the algorithms in detail and apply a system using wireless sensor network.

REFERENCES

[1] S. R. Madden, and M. Franklin, "Fjording the stream: An architecture for queries over streaming sensor data", 18th International Conference on Data Engineering, pp 555-566, 2002.

[2] J. M. Hellerslein, W. Hong, and S. R. Madden, "The Sensor Spectrum: Technology, Trends, and Requirements", ACM SIGMOD Record, Vol. 32, issue 4, pp 22-27, 2003.

[3] K. L. Wu, S. K. Chen, and P. S. Yu, "Interval Query Indexing for Efficient Stream Processing", In Proc. of ACM Int. Conf. on Information and Knowledge Management, pp 88-97, 2004.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems", Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp 1-16, 2002.

[5] L. Golab, and M. T. Özsu, "Issues in Data Stream Management", ACM SIGMOD Record, Vol. 32, No. 2, pp 5-14, 2003.

[6] H. Samet, "Design and Analysis of Spatial Data Structures", Addison-Wesley, 1990.

[7] E. N. Hanson, and M. Chaabouni, "The IBS tree: A data Structure for finding all intervals that overlap a point", Technical Report WSU-CS-90-11, Wright State University, 1990.

[8] E. N. Hanson and T. Johnson, "Selection predicate indexing for active databases using interval skip lists", Information Systems, Vol. 21, No. 3, pp 269-298, 1996.

ACKNOWLEDGEMENT

This research was supported by a grant (#07 국토정보 C05) from Cutting-edge Urban Development – Korean Land Spatialization Research Program funded by Ministry of Construction & Transportation of Korean government.