A Query Index for Stream Data Using Interval Skip Lists Exploiting Locality

Jun-Ki Min

School of Internet-Media Engineering
Korea University of Technology and Education
Byeongcheon-myeon, Cheonan, Chungnam, Republic of Korea, 330-708
jkmin@kut.ac.kr

Abstract. To accelerate the query performance, diverse continuous query index schemes have been proposed for stream data processing systems. In general, a stream query contains the range condition. Thus, by using range conditions, the queries are indexed. In this paper, we propose an efficient range query index scheme QUISIS using a modified Interval Skip Lists to accelerate search time. QUISIS utilizes a locality where a value which will arrive in near future is similar to the current value.

Keywords: Stream Data, Query Index, Locality.

1 Introduction

Stream data management systems may receive huge number of data items from stream data source while large number of simultaneous long-running queries is registered and active[1,2]. In this case, if all registered queries are invoked whenever a stream data item arrives, the system performance degrades. Therefore, Query indexes are built on registered continuous queries [3]. Upon each stream data arrives, a CQ engine searches for matching queries using these indexes.

Existing query indexes simply maintain the all queries based on well known index structures such as the binary search tree and R-tree. However, some application of stream data processing such as stock market and temperature monitoring has a particular property, which is a locality. For example, the temperature in near future will be similar to the current temperature. Therefore, some or all queries which are currently invoked will be reused in the near future. Therefore, the locality of stream data should be considered in the query indexes.

In this paper, we present a range query index scheme, called *QUISIS* (QUery Index for Stream data using Interval Skip lists). Our work is inspired by BMQ-Index [4]. To the best of our knowledge, Interval Skip list [5] is the most efficient structure to search intervals containing a given point. Thus, QUISIS is based on the Interval Skip List in contrast to BMQ-Index. Using a temporal interesting list (TIL), QUISIS efficiently finds out the query set which can evaluate a newly arrived data item. The experimental results confirm that QUISIS is more efficient than the existing query index schemes.

2 Related Work

Some stream data management systems used balanced binary search trees for query indexes [6]. The query index allows to group query conditions combining all selections into a group-filter operator. As shown in Figure 1, a group filter consists of four data structure: a greater-than balanced binary tree, a less-than balanced binary tree, an equality hash-table, and inequality hash table.

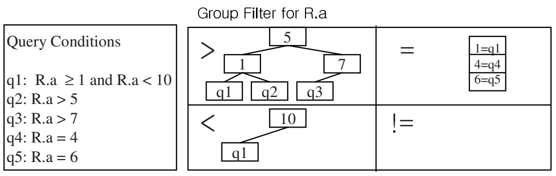


Fig. 1. An example for query indexes using binary search trees

When a data item arrives, balanced binary search trees and hash tables are probed with the value of the tuples. This approach is not appropriate to general range queries which have two bounded conditions. Each bounded condition is indexed in individual binary search tree. Thus, by search of each individual binary tree, unnecessary result may occur.

In addition, for query indexes, multi-dimensional data access methods such as R-Tree [7,8] and grid files can be used [9]. In general, the range conditions of queries are overlapped. These R-tree families are not appropriate for range query indexes since many nodes should be traversed due to a large amount of overlap of query conditions.

Recently, for the range query indexes, BMQ-Index has been proposed. BMQ-Index consists of two data structures: a DMR list, and a stream table. DMR list is a list <DN_1, DN_2, ..., DN_n, DN_{n+1} > of DMR nodes. Let Q = {q_i} be a set of queries. A DMR node DN_j is a tuple <DR_j, +DQSet, -DQSet>. DR_j is a matching region (b_{j-1}, b_j). +DQSet is the set of queries q_k such that $\mathrm{l}_k = \mathrm{b}_{j-1}$ for each selection region (l_k, u_k) of Q_k. -DQSet is the set of queries q_k such that $\mathrm{u}_k = \mathrm{b}_{j-1}$ for each selection region (l_k, u_k) of q_k. Figure 2 shows an example of BMQ-Index. A stream table keeps the recently accessed DMR node.

Let QSet(t) be a set of queries for data v_t at time t and v_t be in the DN_j, and v_{t+1} is in the DN_h i.e., $\mathrm{b}_{j-1} \leq \mathrm{v}_t < b_j$ and $\mathrm{b}_{h-1} \leq \mathrm{v}_{t+1} < \mathrm{b}_h$. QSet(t+1) can be derived as follows:

$$\begin{array}{ll} if & j < h, QSet(t+1) = QSet(t) \cup [\bigcup_{i=j+1}^{h} +DQSet_i] - [\bigcup_{i=j+1}^{h} -DQSet_i] \\ if & j > h, QSet(t+1) = QSet(t) \cup [\bigcup_{i=j}^{h+1} -DQSet_i] - [\bigcup_{i=j}^{h+1} +DQSet_i] \\ if & j = h, QSet(t+1) = QSet(t) \end{array} \quad (1)$$

The authors of BMQ-Index insist that only a small number of DRN nodes is retrieved, if the forthcoming data is not in the region due to the data locality.

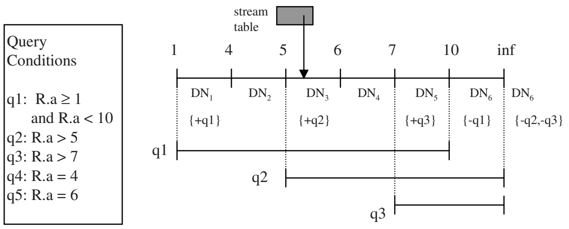


Fig. 2. An example of a BMQ-Index

However, BMQ-Index has some problem. First, if the forthcoming data is quite different from the current data, many DRN nodes should be retrieved like a linear search fashion. Second, BMQ-Index supports only (l, u) style conditions but does not support general condition such as [l,u] and (l, u]. Thus, as shown in Figure 2, q4 and q5 is not registered in BMQ-Index. In addition, BMQ-Index does not work correctly on boundary conditions. For example, if v_t is 5.5, the QSet(t) is {q1,q2}. Then, if v_{t+1} is 5, QSet(t+1) is also {q1,q2} by the above equation. However, the actual query set for v_{t+1} is q1.

3 QUISIS

In this section, we present the details of our proposed approach, QUISIS. As mentioned earlier, QUISIS is based on Interval Skip Lists[5]. Thus, we first introduce Interval Skip Lists, and then present our scheme.

3.1 Interval Skip Lists

Interval Skip Lists are similar to linked lists, except each node in the list has one or more forward pointers. The number of forward pointers of the node is called the level of the node. The level of a node is chosen at random. The probability a new node has k level is:

$$P(k) = \begin{cases} 0 & \text{for} k < 1 \\ (1-p) \cdot p^{k-1} & \text{for} k \geq 1 \end{cases} \tag{2}$$

With p = 1/2, the distribution node levels will allocate approximately 1/2 of the nodes with one forward pointer, 1/4 with two forward pointers, and so on. A node's forward pointer at level l points to the next node with greater that or equal to l level.

In addition, nodes and forward pointers have markers in order to indicate the corresponding intervals. Consider I = (A,B) to be indexed. End points A and B are inserted in the list as nodes. Consider some forward edges from a node with value X to a node with value Y (i.e., X < Y). A marker containing the identifier of I will be placed on edge (X,Y) if and only if the following conditions hold:

- containment: I contains the interval (X,Y)
- maximality: There is no forward pointer in the list corresponding to an interval (X', Y') that lies within I and that contains (X,Y).

In addition, if a marker for I is placed on an edge, then the nodes of that edge and have a value contained in I will also have a marker (called eqMarker) placed on them for I.

The time complexity of Interval Skip Lists is known as O(log N) where N is the number of intervals. Since we present the extended version of the search algorithm in Section 3.2, we omit the formal description of the search algorithm for Interval Skip Lists.

3.2 Behavior of QUISIS

In Figure 3, QUISIS is shown when the current data item is 5.5. Given search key, the search procedure starts from Header in Interval Skip Lists. In stream data environment, a locality such that a data in the near future is similar to the current data occurs. By using this property, we devise the QUISIS based on Interval Skip Lists.

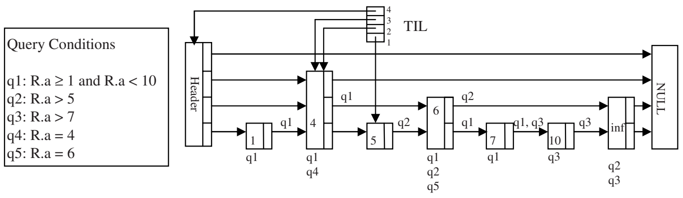


Fig. 3. An example of QUISIS

In order to keep the visited edges by the current data item, a temporal interesting list (TIL) is used. TIL records the nodes with level from MAX level to 1 whose forward pointer with level l represents an interval contains the current data item. As shown in Figure 3, the interval [5,6) represented by the node pointed by TIL with level 1 contains the current data item 5.5.

In Figure 3, we can interpret *Header* and *Null* such that *Header* represents the smallest value and *Null* represents the largest value. So *Header* is smaller than $-\infty$ and *Null* is greater than ∞ in contrast to the conventional number system. Thus, the intervals represented by the nodes in TIL have the property such that:

Property 1. The interval by TIL with level i is contained in the interval by TIL with level $i + 1$.

For example, [5,6) by TIL with level 1 is contained in [4,6) by TIL with level 2 and also is contained in [4,*Null*) by TIL with level 3. By using this property, QUISIS reduces the search space efficiently compared with original Interval Skip Lists.

```
QSet //a query set for previous key
TIL // a list points to the nodes of QUISIS
**Procedure** findQuery(*key* )
**begin**
1.  **if**(TIL[1]->value = *key*) {
2.      **for**( i = TIL[1]->level; i $\geq$ 1; i−−) QSet := QSet - TIL[1]->markers[i]
3.      QSet := QSet $\cup$ TIL[1]->eqMarker
4.  } **else if**(TIL[1]->value < *key* **and**
          (TIL[1]->forward[1] = NULL **or** *key* < TIL[1]->forward[1]->key)) {
5.      QSet := QSet - TIL[1]->eqMarker
6.      **for**(i = TIL[1]->level; i $\geq$1; i–) QSet := QSet $\cup$ TIL[1]->markers[i]
7.  } **else** {
8.      QSet := QSet - TIL[1]->eqMarkers
9.      **if**(TIL[1]->forward[1] = NULL **or** *key* $\geq$ TIL[1]->forward[1]->value ) {
10.         **for**(i := 1; i $\leq$ maxLevel ; i++)
11.             **if**(TIL[i]->forward[i] = NULL **or** *key* < TIL[i]->forward[i]->value) break
12.             **else** QSet = QSet - TIL[i]->markers[i]
13.     } **else** {
14.         **for**(i = 1; i $\leq$ maxLevel; i++)
15.             **if**(TIL[i]= Header **and** *key* $\geq$ TIL[i]->value) break
16.             **else** QSet := QSet - TIL[i]->markers[i]
17.     }
18.     anode := TIL[−−i]
19.     **while**(i $\geq$ 1) {
20.         **while**(anode->forward[i] $\neq$ NULL **and** anode->forward[i]->value *le key*)
                anode = anode->forward[i]
21.         **if**(anode $\neq$ Header and anode->value $\neq$ *key*) QSet := QSet $\cup$ anode->markers[i]
22.         **else if**(anode $\neq$ Header) QSet := QSet $\cup$ anode->eqMarker[i]
23.         TIL[i] = anode;
24.         i:= i-1
25.     }
26. }
27. return QSet
**end**
```

Fig. 4. An algorithm of the event handler for `endElement`

In order to exploit TIL, we devised the procedure findQuery() using the Property 1. An outline of an implementation of findQuery() is shown in Figure 4.

Basically, the behavior of the procedure findQuery() is changed according to the condition of newly arrived data value (i.e., *key*) and the interval $[v_1, u_1)$ by TIL with level 1.

If *key* is equal to v_1 of the node n pointed by TIL with level 1 (Line 1-3 in Figure 4), all markers on the edges starting from n are removed (Line 2) since QSet may contain queries whose intervals are (v_1, -). Instead, eqMarker of n is added (Line 3) since eqMarker of n contains the queries whose interval contains v_1 and the queries in eqMarker are on that edges which are ended or started at n. For example, when the data was 5.5, QSet was {q1,q2} and a new data item 5 arrives, the TIL[1] is the node 5. Thus, {q2} is removed from QSet by Line 2. And since eqMarker of the node 5 is $\emptyset$, the final query result is {q1}.

If *key* is in (v_1, u_1)(Line 4-6 in Figure 4), the queries in eqMarker of n are removed since the QSet may contain queries whose intervals are (-, v_1]. Instead, all markers on the edges starting from n are added.

If *key* is not in (v_1,u_1) (Line 7-27 in Figure 4), the procedure looks for the interval $[v_i, u_i)$ by TIL with level i which contains *key* (Line 8-17). This step is separated into two cases: key $\geq u_1$ (Line 9-12) and key $< v_1$ (Line (Line 13-17). Also, in this step, markers on the edges with level from 1 to i-1 are removed from QSet (Line 12 and 16). And then, the procedure gathers queries starting from the node (i.e., *anode*) whose value is v_i (Line 19-25). In this case, since the marker on the edge of *anode* with level i is already in QSet, level i decreases (Line 18).

If the interval represented by a forward pointer of *anode* with level i does not contain key, a search procedure traverses to the next node pointed by the forward pointer of a node with a level i (Line 21). If the value of *anode* is equal to key, eqMarker of *anode* is added (Line 22). Otherwise the marker on the forward pointer is added (Line 21). Then, the *anode* is set to TIL[i](Line 23) and i is dropped into $i-1$(Line 24). The search procedure continues until the level l is to be 1.

For example, when the data was 5.5, QSet was {q1,q2} and a new data item 13 arrives, [4, $Null$) represented by TIL[3] contains 13. Therefore, the procedure removes the search overhead starting from $Header$. {q2} and {q1} which are markers of TIL[1] and TIL[2], respectively, are removed from QSet (Line 9-12). Then, the procedure gathers queries starting from the node 4 with level 2 (Line 18). Since [4,6) does not contain 13, the procedure looks for next interval [6, $Null$) represented by node 6 with level 2. Since 13 is in [6, $Null$) but not equal to 6, a marker q2 is added. And TIL[2] points the node 6. Then, the procedure searches the list from the node 6 with level 1. Since [10, inf) contains 13, a marker {q3} is added. Finally, QSet is {q2, q3}.

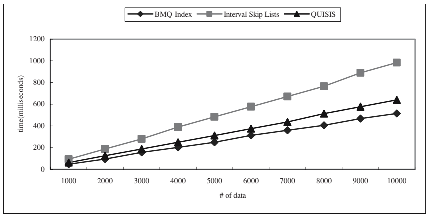
In aspect of using the data locality, our proposed scheme QUISIS is similar to BMQ-Index. However, since QUISIS is based on Interval Skip Lists, QUISIS is much more efficient than BMQ-Index in general cases. Our experiments demonstrate the efficiency of QUISIS.

4 Experiments

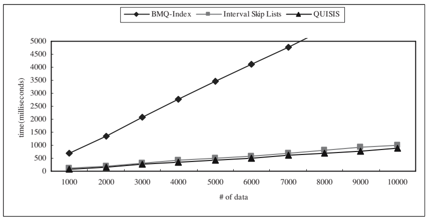
In this section, we show the efficiency of QUISIS compared with the diverse query index techniques: BMQ-Index and Interval Skip Lists. The experiment performed on Windows-XP machine with a Pentium IV-3.0Ghz CPU and 1GB RAM. We evaluated the performance of QUISIS using the synthetic data over various parameters. We implemented a revised version of BMQ-Index which works correctly on the boundary conditions. The default experimental environment is summarized in Table 1. In Table 1, length of query range (W) denotes the average length of query condition normalized by the attribute domain and fluctuation level (FL) denotes the average distance of two consecutive data normalized by the attribute domain. Therefore, as FL decrease, the locality appears severely.

Table 1. Table of Symbols

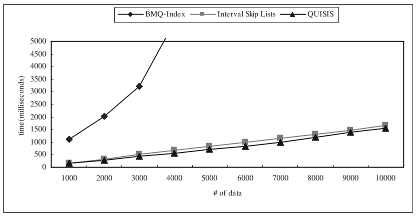
Parameter	value
Attribute domain	$1 \sim 1{,}000{,}000$
# of Queries	100,000
Length of query range (W)	0.1% (= 1,000)
# of Data	$1{,}000 \sim 10{,}000$
Fluctuation level (FL)	0.01% (= 100)



(a) FL = 0.01%



(b) FL = 0.1%



(c) FL = 1%

Fig. 5. The results with varying the number of data

We empirically performed experiments with varying parameters. However, due to the space limitation, we show only the experimental result when values of FL are 0.01%, 0.1% and 1%.

Our proposed index scheme QUISIS shows the best performance except the case when FL is 0.01%. Figure 5-(a), BMQ-Index shows the best performance when FL is 0.01% (i.e., high locality) due to its simple structure. In BMQ-Index, if the forthcoming data is different from the current data, many DMR nodes should be retrieved. Therefore, BMQ-Index shows the worst performance when FL is 0.1% (see Figure 5-(b)) and 1% (see Figure 5-(c)). In other words, BMQ-Index only fits on the high locality cases. In contrast to BMQ-Index, QUISIS shows good performance over all cases since QUISIS efficiently retrieves the query set using TIL and removes the overhead searching from *Header*. The performance of Interval Skip Lists does not affected by FL. As shown in Figure 5-(c), Interval Skip Lists shows the good performance when FL = 1.0%. Particulary, when FL are 0.1% and 1%, Interval Skip Lists is superior to BMQ-Index.

Consequently, QUISIS is shown to provide reasonable performance over diverse data locality.

5 Conclusion

In this paper, we present an efficient scheme for query indexing, called QUISIS which utilizes the data locality. QUISIS is based on Interval Skip Lists. In order to maintain the current locality, TIL (temporal interesting list) is equipped. To show the efficiency of QUISIS, we conducted an extensive experimental study with the synthetic data. The experimental results demonstrate that QUISIS is superior to existing query index schemes.

References

1. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., Widom, J.: STREAM: The Stanford Stream Data Manager. IEEE Data Engineering Bulletin **26** (2003)
2. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Reiss, F., Shah, M.A.: TelegraphCQ: Continuous Dataflow Processing. In: ACM SIGMOD Conference. (2003)
3. Ross, K.A.: Conjunctive selection conditions in main memory. In: PODS Conference. (2002)
4. Lee, J., Lee, Y., Kang, S., Jin, H., Lee, S., Kim, B., Song, J.: BMQ-Index: Shared and Incremental Processing of Border Monitoring Queries over Data Streams. In: International Conference on Mobile Data Management (MDM'06). (2006)
5. Hanson, E.N., Johnson, T.: Selection Predicate Indexing for Active Databases Using Interval Skip Lists. Information Systems **21** (1996)
6. Madden, S., Shah, M.A., Hellerstein, J.M., Raman, V.: Continuously adaptive continuous queries over streams. In: ACM SIGMOD Conference. (2002)
7. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: ACM SIGMOD Conference. (1984)
8. Brinkhoff, T., Kriegel, H., Scheneider, R., Seeger, B.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: ACM SIGMOD Conference. (1990)
9. Choi, S., Lee, J., Kim, S.M., Jo, S., Song, J., Lee, Y.J.: Accelerating Database Processing at e-Commerce Sites. In: International Conference on Electronic Commerce and Web Technologies. (2004)