Query Optimum Scheduling Strategy on Network Security Monitoring Data Streams

Lina Wang and Ying Ren
Naval Aviation University, Shandong Yantai, China,
Email: wanglina5677@163.com; ren.ying@163.com

Abstract. This thesis studies on the query mechanism of network security monitoring data streams, and researches storage flow data in memory adopting a method of using flow data as materialized sharing intermediate results of compression. Through the experiments of two kinds of flow data storage structure of StreamQCTree-D and QC-Tree, confirmed the dynamic scheme of StreamQCTree relative to QC-Tree in the lower part of the query performance to get higher data compression rate, further validation of the data compression of multiple query optimization is effective.

1. Introduction
In recent years, the application field of information processing technology has been widely expanded, and there are more and more threats to network security. It is of great practical significance to monitor the network and analyze the monitoring data for the purpose of security. Network security monitoring produces massive, sustained and fast data. Data stream processing technology can continuously and quickly analyze real-time data, so data stream processing technology has become another hot spot in database research field[1].

Large scale network security monitoring flow data processing system, when managing and analyzing security monitoring, multiple systems will log into the system at the same time, and produce a large number of queries, and each query is handled separately. This query method is inefficient [2]. Multiple queries may share the same sub tasks. Through multi query optimization, the efficiency of user query can be improved and the computation cost of system can be reduced. Multi query optimization generally adopts two methods: optimizing query plan and materializing sharing intermediate results. In this paper, an in-depth study of query processing mechanism in data stream is carried out[3]. A method is proposed, which uses stream data square as the intermediate result of materialized sharing, to study the compressed storage of stream data in memory, optimize the storage efficiency of the system in order to save more intermediate results.

2. Data flow multiple query optimization
A given query set \( Q = \{Q_1, ..., Q_q\} \) consists of complex SQL queries and contains relational sets \( Q_1, ..., Q_q \). Each query \( Q_i \) is a subset of the set of relational attributes. The number of elements in the representation of a relationship is expressed. The data stream query processing engine only allows sequential and single sweep of data flow tables \( R_1, ..., R_r \), and cannot be read again. The multi query optimization technology is used to share the intermediate results of materialized sharing, and the results of multiple query shared by multiple queries are cached in memory so that multiple queries can be read for many times. It is suitable for the data flow to scan the characteristics of the data in a single pass and improve the efficiency of the query. The multi query optimization technology of
materialization sharing intermediate results is shown in Figure 1.

```
| Intermediate results |
|----------------------|
| DS₁ of physicochemical |
```

![Figure 1. Multi query optimization of data flow](image)

Compressed materialized intermediate results are stored in memory for reuse. With the increase of query time window, storage space is increased. Limited memory space can not meet the increasing storage requirements, and the method of compressing and sharing the intermediate results can solve this problem better.

When all queries in a multi query set are limited to aggregate queries, the materialized intermediate results can be mapped to partial stream data parties. A better flow data compression method should satisfy three conditions as far as possible:

- removing data square redundancy during compression.
- there is no need to unzip the query.
- targeted pre queries on data streams.

### 3. Stream data square compression structure

Selecting the appropriate stream data side compression structure can save more information in the same capacity memory. At present, there are a variety of compression structures such as QC-tree⁴, Dwarf⁵, etc. these compressed structures are all data oriented static data. The flow data of the dynamic data is stored in the memory, and the dynamic data is persistent and infinite. It can not hold the completely materialized data square compression structure in the limited memory space, and can only store the stream data slice for a certain time. In order to save more stream data slices in memory, it is necessary to reduce the data. The capacity of a small single slice.

In this paper, the implementation framework of data stream data square for data flow system is proposed for real-time data flow analysis, and the data square structure of the compressed stream, StreamQC-Tree⁶, and its update and query algorithm are implemented.

The definition of StreamQC-Tree StreamQC-Tree is similar to QC-Tree, except for the following properties:

- StreamQC-Tree, the root node root directly connects each time dimension node with the edge. Each time dimension node.It includes a QC-Tree subtree, which corresponds to data square time slice $SC_{u}$.  

• StreamQCTree contains all basic upper bound classes, and select Add part to add the last class.
• in StreamQCTree, each node contains an additional value \(t\cos\), which represents the cost of the retrieval node. For things the node cost is 1, and the non materialized node is the time to access its subsequent nodes when aggregating queries.
• all non materialized nodes do not store their metrics.

4. Selection of materialized nodes
The definition of 2 materialized node selection problem is given a database schema \(R\), constraint condition \(T\), query set \(Q\) and cost evaluation function \(C\). The selection problem of physical and chemical nodes is to select a materialized node set \(V\) on \(R\), making the cost minimization of the materialized node \(C(R,V,Q)\) in \(V\) under the condition of conditional \(T\).

4.1. Cost calculation
A data cube has \(D\) dimension. Data parties store and allocate \(m\) storage units, data side time windows \(W\), and data flow slide window \(s\). Each data square time slice assigns memory space \(M_{sc} = m \times s / W\). Set up the basic upper class to account for the memory \(M_{bas}\). Additional upper bound classes can be used as: \(M_{add} = M_{sc} - M_{bas}\).

Set pointer storage to account for \(P\), dimension mark store account for \(La\), measure value store \(Me\), and its subsequent node number is cost. If an additional upper class is added, the storage space will be increased.

\[
M_{aud} = \cos t \times P + Me + x \times La
\]  

Where \(X\) represents the dimension mark of the upper bound, it is determined that the value is generally the number of dimension tags after the first * in the dimension tag sequence of the upper bound \(0 \leq x < |D|\).

4.2. Income calculation
For any additional upper bound \(UB_i\), if the probability \(UB_i\) of nodes \(ubi\) in the tree is accessed in the query set, the number of materialized visits \(C_{um}(UB_i)\) is that the number of times that is materialized is \(C_{m}(UB_i)\). The yield of the upper bound of materialization is as follows:

\[
B_{ubi} = P_{ubi} \times (C_{um}(UB_i) - C_{m}(UB_i)) / M_{aud}
\]  

It is a pair of ratio \(B_{ubi}\), which can evaluate the physical and chemical gains relative to the other upper bounds, to evaluate and select the best additional upper class materialization to achieve the lowest average query response time.

5. The update and query of data square of 4 compressed flow

5.1. StreamQCTree update algorithm
StreamQCTree uses the time slice data square model. In the update algorithm, the data stream is constantly arriving, because each time \(\Delta t\) interval will generate a QC-Tree subtree of the data square time slice. Updating only needs to add the Ti time subtree to the Root node in StreamQCTree.

There is only one case of deletion, when the time slice of Ti data is expired in memory, the sub tree is deleted from memory. Simply release the memory of the QC-Tree subtree and remove the pointer from Root to the subtree.
5.2. StreamQCTree query algorithm

Function traverBUB (sqt, i, s[[]]

For each BUB tag in the sqttree layer i layer
For from the i+1 layer in the tree to the |D| layer
  traver(sqt, s[i+1]);
if i==|D|
  result'=getresult();
  result=fun(result');
Return result.

Query algorithm is similar to QC-Tree query algorithm, because some nodes are not materialized, so the query involves non materialization.

Query exceptions are produced when the nodes are nodes. After locating the query exception, according to the tree level i of the exception, we determine the basic upper class node in the tree to be accessed by the i-1 layer, that is, function traverBUB ()

6. Experiment

In the experiment, the algorithm is implemented by VC++6.0. The running environment is PC installed with Windows XP Professional, and the hardware is configured as 160G hard disk and 2G memory. It is assumed that the tuples on the data flow are evenly distributed on the time dimension. The data has 6 dimensions, each dimension is 100, the data square time window W=10, Zipf (factor=1.2), each time has 100k new yuan group arrival, the total allocation of 200M memory space, each time slice allocates the memory 10M. The experiment uses two stream data storage structures StreamQCTree-D and QC-Tree, and StreamQCTree-D adopts dynamic selection scheme StreamQCTree.

The experimental results verify the effectiveness of the StreamQCTree compression method. As shown in Figure 2, StreamQCTree further compresses the memory occupancy space compared to QC-Tree. StreamQCTree-D uses a dynamic selection scheme to select part of the tree node, which materialize the nodes in the tree with the highest frequency, and reduce the query frequency as much as possible or do not query the tree nodes. The response time gap between the query response time of the QC-Tree is smaller and the smaller query performance is reduced, as shown in Figure 3.

7. Conclusion

In this paper, a method of sharing intermediate results with streaming data side as materialized sharing is adopted. Experiments are carried out through two kinds of stream data square storage structure StreamQCTree-D and QC-Tree. The experimental results verify that the StreamQCTree relative QC-Tree using dynamic materialization scheme obtains higher data compression rate in reducing part query performance, and further validates the data compression rate. The effectiveness of data
compression for multi query optimization is verified.

8. References

[1] Chien J T, Wu C C. Discriminant Waveletfaces and Nearest Feature Classifiers for Face Recognition[J]. IEEE Trans. on PAMI, 2002, 24(12): 1644-1649.

[2] Zheng Wen-ming, Zou Cai-rong, Zhao Li. Face Recognition Using Two Novel Nearest Neighbor Classifiers[C]//Proceedings of ICASSP. 2004: 725-728.

[3] Prasan Roy, S. Seshadri, S. Sudarshan, et al. Efficient and Extensible Algorithms for Multi Query Optimization[C]//Proceedings of the 19th ACM SIGMOD International Conference on Management of Data, 2000, Dallas, USA: ACM Press: 249-260.

[4] Laks V. S. Lakshmanan, Jian Pei, Yan Zhan. QC-trees: An efficient summary structure for semantic OLAP[C]. //Proc of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, California: ACM: 64-75.

[5] Yannis Sismanis, Antonios Deligiannakis, Nick Roussopoulos, et al. Dwarf: Shrinking the petacube[C]//Proc of the 2002 ACM-SIGMOD international conference management of data, 2002, Madison, Wisconsin: ACM: 464-475.