[1]H*-tree: 데이터 스트림의 다차원 분석을 위한 개선된 데이터 큐브 구조

심상예*, 정우상*, 이연*, 신숭선*, 이동욱*, 배해영*
*인하대학교 컴퓨터정보공학과
e-mail : xiangrui_chen@yahoo.cn, cyxyuxiang@gmail.com,
{leeyeon,hermit,dwlee}@dblab.inha.ac.kr, hybae@inha.ac.kr

[1]H*-tree: An Improved Data Cube Structure for Multi-dimensional Analysis of Data Streams

XiangRui Chen*, YuXiang Cheng*, Yan Li*, Song-Sun Shin*, Dong-Wook Lee*, Hae-Young Bae*
*Dept. of Computer Science and Information Engineering, Inha University

요 약(Abstract)

In this paper, based on H-tree, which is proposed as the basic data cube structure for multi-dimensional data stream analysis, we have done some analysis. We find there are a lot of redundant nodes in H-tree, and the tree-build method can be improved for saving not only memory, but also time used for inserting tuples. Also, to facilitate more fast and large amount of data stream analysis, which is very important for stream research, H*-tree is designed and developed. Our performance study compare the proposed H*-tree and H-tree, identify that H*-tree can save more memory and time during inserting data stream tuples.

1. Introduction

Nowadays, in the real-time surveillance systems, telecommunication systems, and other application areas, anomaly detecting is more important than high level summary, which is related to the data stream on-line analysis. As these environments always generate tremendous (potentially infinite) amount of data stream [7], the data structure for on-line, multi-dimensional analysis of data stream becomes a challenging task. Similar to the role of data cube architecture [1] in the analysis of data warehousing and OLAP technology [9], stream cube architecture [3] is proposed as the architecture for multi-dimensional analysis of data streams, especially for anomaly detecting and unusual patterns mining[8].

Most data streams are at low-level or multi-dimensional in nature, and it requires more multi-level (ML) / multi-dimension (MD) processing. Meet the new requirements of stream OLAP, as referred in [4], stream cube belongs to the concept of data cube, but it is a kind of selectively materialized cube, which uses H-tree structure to store computed cells. It has 3 significant features [5]: (1) tilted time frame, (2) two critical layers: a minimal interesting layer and an observation layer, and (3) partial computation of data cubes by popular-path cubing. The stream data cubes so constructed are much smaller than those constructed from the raw stream data but will still be effective for multi-dimensional stream data analysis tasks. It has been successfully implemented in the MAIDS [6] project, and it is proved that H-tree is still the most appropriate structure for data stream since most other structures need to either scan data sets more than once or know the sparse or dense parts beforehand, which does not fit the single-scan and dynamic nature of multi-dimensional data stream.

In this paper, Section 2 presents the H-tree structure and lists 3 potentially big problems for research if the stream data is very fast, with the growing size of stream cube resort in memory. Based on this, in Section 3, we present an improved data structure, H*-tree, which reserves the strong point of H-tree, meets the new requirements and the problems analyzed in Section 2. In Section 4, we give the performance study by comparing the memory and time used for inserting growing large scale of data sets with H-tree. At last, our previous study is concluded in Section 5.

2. Related work

In this section, we introduce the concepts related to H-tree, define the problems after examining of H-tree.

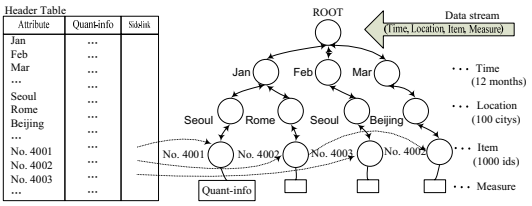
2.1 H-tree definition

As introduced in the Section 1, H-tree is used to store the computed cells of stream cube. It is a hyper-tree structure, and an example can be built like Figure 1.

As lack of space, we omit the tree-build steps, which you may find in [4]. What we need to know, H-tree is the most basic structure for stream cube as every coming multi-dimensional data stream tuple should be inserted into the tree by single-scan method first. So the efficiency of tuple insert strategy should be considered more detailedly. What's more important, many other operations are based on the data

[1] **This research was supported by a grant (07KLSGC05) from Cutting-edge Urban Development - Korean Land Spatialization Research Project funded by Ministry of Construction & Transportation of Korean government.**

structure of H-tree, e.g. H-cubing [4]. It means that the designing of H-tree should be more carefully. We examined H-tree and find there are 3 problems which can be improved as follows.



(Figure 1) H-tree structure

2.2 Problems existed in H-tree

Although the novel features of H-tree [4], construction cost, completeness and compactness can fulfill the emergency of unbounded data stream. But there are still some improvements can be implemented.

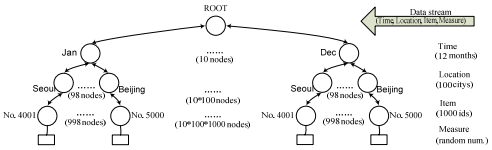
2.2.1 Potentially redundant nodes

Review Figure 1, examining the tree-build method in [4], we give a tuples insert example. Since the first tuple, (Jan, Seoul, No. 4001, 200), is inserted into the H-tree, it creates 3 nodes and the num 200 is stored in Quant-info in the leaf. If a tuple as (Jan, Seoul, No. 4001, 100) is comes, it can share the path built for the first tuple, saving memory in a way (the whole tree resides in memory), but if a third tuple, (Feb, Seoul, No. 4003, 300) arrives, it will create another node labeled 'Seoul' in the second layer (we take the root as 0 layer) in the tree. As a result, in the second layer, the second 'Seoul' node can be considered as a redundant node if the first created one can be shared when we redesign the tree structure. What's more, considering the preprocessed data stream which will arrive the stream cube, the cardinality of the first attribute 'Time' is 12 (months), the cardinality of the second attribute 'Location' is 100 (cities), and the cardinality of the third attribute 'Item' is 1000 (product ids). Reasoning following H-tree structure, for 1000+x tuples, there are at least x redundant nodes in the second layer. As the layers (dimensions) in H-tree are designed in cardinality-ascending order, if data stream tuple contains more attributes, which means that there are more layers in the H-tree. Considering the cardinality of every attribute, the deeper the tree will be, the more redundant nodes there are in the tree.

H-tree is stored in the limited memory for stream cube, with the growing size of data stream arrived, more and more redundant nodes will waste the memory to a large extent. Although the problem is not considerable when the data size is not very large, in real environments, redesigning the tree structure is still necessary. We will give a new tree structure, H*-tree in Section 3.

2.2.2 Delaying of on-line analyzing

H-tree is constructed dynamically and real-timely [4], it means that the detailed H-tree building process totally depends on the data stream received. It seems that this method can save memory in a way. Is it true? In our opinion, it is only 'true' to limited extent, when the data size is not very large. As the data stream for analysis is always continuously and unboundedly, considering the worst case, the complete tree, as example shown in Figure 2, will be finally built during the analyzing period. Then, the problem comes forth, for anomaly detecting and other on-line analysis of data stream, the time and memory spent on creating H-tree during analyzing period will affect the result quality of analysis to some extent. For example, the result maybe not very accurate, or the anomaly will not be detected in time because of delaying of creating tree [8]. So when to build the tree? How to make the analysis result more exactly and in time, especially for anomaly detecting? We will answer these questions in Section 3.



(Figure 2) Complete H-tree structure

2.2.3 The randomly layout of layers

As discussed above, H-tree is constructed dynamically and real-timely. The layout of every layer is randomly, according to the order of tuples arrived. Reviewing the insert method of a new tuple [4], take (Jan, Rome, No. 4052, 265) for example, it will first retrieve the existed nodes in the first layer, searching for the node labeled 'Jan'. If returns 'true', similarly, it will retrieve the existed nodes in the second layer, searching for the node labeled 'Rome', recursive like this in every layer.

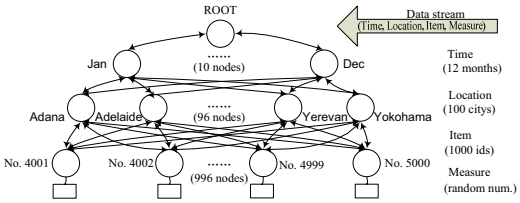
Examining this inserting method, focus on the retrieving of existed nodes at every level, there are some improvements can be done. Take the nodes existed in a level as a list of numbers, if we want to retrieve a number equals x, which kind of layouts maybe better, randomly or orderly? For randomly layout, 'sequence search' is the only option for best performance. But for orderly layout, 'binary search' is the best choice as the time used will be much shorter compared with any other method, including 'sequence search'. Then, it comes forth the question, is it possible to make the layout of every layer in H-tree orderly? Maybe it is possible. We have not proved the possibility, but if possible, it will be very difficult because H-tree is constructed dynamically and real-timely. In Section 3, we will present another data structure, H*-tree, the layout of every layer in which is orderly.

3. H*-tree definition

In this section, we solve the problems defined in Section 2 and present an improved data structure, H*-tree based on H-tree, which will be more cost-effective for analyzing multi-dimensional data streams.

In the real environment, as the tree used in stream cube is constructed in a cardinality-ascending order, the row data stream must be formatted to the tuple that will insert into the tree. As we can define the format before analyzing, we know how many attributes there will be in one tuple, and it equals the number of levels in the tree. Also, we know the cardinality of every attribute in the tuple. With this kind of important information, we can create the complete (Figure 2) tree directly before analyzing, answering the question in

2.2.2. But as discussed before in 2.2.1, constructing a complete H-tree will no doubt contain too many redundant nodes, which waste a large scale of limited memory, and the waste size will grow up on the fly with the growing of the attributes number in a tuple. What's more, considering the problem presented in 2.2.3, the insert time will also grow up a lot. So, redesigning the tree structure and share nodes in the same layer is greatly needed, and the H*-tree we designed is as follows in Figure 3.



(Figure 3) Complete H*-tree

As shown above, in the complete H*-tree example, there are 1+12+100+1000=1113 nodes. But in the complete H-tree example (Figure 2), totally, there are 1 + 1*12 + 1*12*100 + 1*12*100*1000 = 1201213 nodes, compared with H*tree, there are 1200100 redundant nodes, which is more than 1000 times of the really needed nodes. Also, compared with Figure 1, there will be no Header Table or side-link in memory, which will also save the limited resource in a way.

Besides, as described in 2.2.3, we can make the layout of every layer in H*-tree structure orderly during tree building period, which will make the insertion of data stream tuples more efficiently comparing with H-tree.

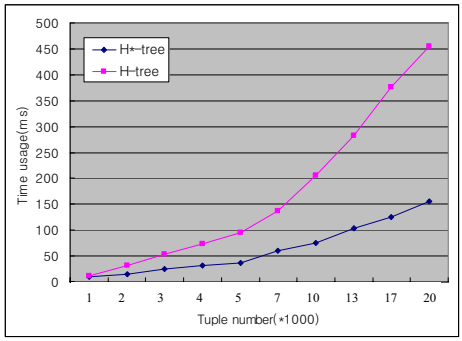
4. Performance study

In this section, we implement the H-tree proposed in [4] and H*-tree proposed in Section 3. By inserting data tuples continuously, we give the performance study of them. The data sets used for test is dynamically generated by the data generator we designed, and for better testing, we specify 10 different tuple sizes from 1,000 to 20,000 and tested 10 times.

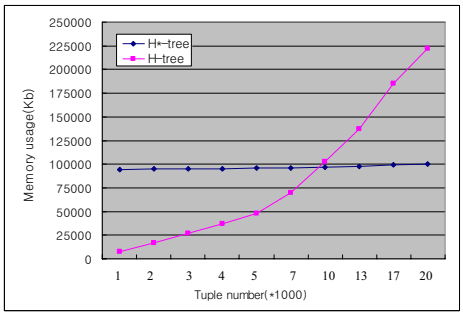
All experiments were conduced on a 1.8 GHz Pentium 4 PC with 1.25 GB main memory, running Microsoft-XP Professional. All the methods were implemented using Microsoft Visual C++ 6.0.

The performance results of H*-tree and H-tree are reported by recording the memory and time usage for inserting tuples as following analysis, which is the very important angles for the considering and designing of data structure used in stream cube.

Figure 4 shows the processing time usage for inserting tuples, with the increasing size of the data tuples. Since H-tree creates tree dynamically and the layout of every layer is randomly, which is not good for retrieval nodes existed in the layer, the total insert time of H-tree is much higher than H*-tree. With the growing of data tuples amount, the gap between two trees is more and more wide, which can be seen from the trend of the curves. Also, Figure 5 shows the processing memory usage for inserting tuples, with the increasing size of the data tuples. As a result of H-tree's limitation presented in Section 2, the size of memory used grows rapidly with the growing of tuple size. To the opposite, as there is no redundant nodes in H*-tree, the memory usage curve is steady and not very high.



(Figure 4) Time usage for tuples insertion



(Figure 5) Memory usage for tuples insertion

What need to be further discussed, shown in Figure 5, two curves have a point of intersection. As we have explained in 2.2.1, according to the data structure of H-tree, the dynamically tree-build method can save memory in a way, when the tuple size arrived is not very large (under 10,000 in Figure 5, if the PC performance is better, maybe it can be much higher), and the number of H-tree nodes existed in memory will not be very big, to the opposite, as we designed in H*-tree, we will build the complete tree in memory before inserting tuples, all the H*-tree nodes will exist from the very beginning of the test. Then, it will use more memory than H-tree at first. But obviously, with the growing of tuples number, there will be more and more nodes created in H-tree (contains more redundant nodes), and no 'new' node in H*-tree. As a result of this, after a threshold, H-tree will hold more memory compared with H*-tree, the memory usage curve will also be not stable and in a much more fast growing trend, comparing with H*-tree.

5. Conclusions

In this paper, we have analyzed the H-tree data structure used in stream cube [4], and proposed a more feasible and cost-effective data structure, H*-tree. By performance study, comparing the memory and time usage between H*-tree and

H-tree, we confirmed that potentially large amount of redundant node is the essential and direct reason why the memory usage curve of H-tree grows so fast, and dynamically tree-build method, randomly layout of tree-layer are the essential reasons why the time usage curve of H-tree is much high than H*-tree. All these problems can be solved using the improved tree structure, H*-tree.

Now, we are sure that the new data structure, H*-tree, is a better choice for on-line analyzing of multi-dimensional data stream. And in the future, more research works related to the multi-dimensional analysis of data stream can based on, H*-tree, the fundamental data structure of stream cube.

Reference

[1] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, “Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals,” Data Mining and Knowledge Discovery, vol. 1, pp. 29-54, 1997.

[2] J. Han, J. Pei, G. Dong, and K. Wang, “Efficient Computation of Iceberg Cubes With Complex Measures,” Proc. ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '01), pp. 1-12, May 2001.

[3] Y. Chen, G. Dong, J. Han, B. Wah & J. Wang, “Multi-dimensional Regression Analysis of Time-series Data Streams,” VLDB 2002.

[4] Y. Chen, G. Dong, J. Han, J. Pei, B. W. Wu, and J. Wang. Online analytical processing stream data: Is it feasible? [C]. DMKD 2002.

[5] J. Han, Y. Chen, G. Dong, J. Pei, B.W. Wah, J. Wang, and D. Cai, “Stream Cube: An Architecture for Multi-Dimensional Analysis of Data Streams,” Distributed and Parallel Databases J., 2005.

[6] Y. Dora Cai, D. Clutter, G. Pape, J. Han, M. Welge, L. Auvil. “MAIDS: Mining Alarming Incidents from Data Streams,” ACM-SIGMOD Int'l Conf. Management of Data (SIGMOD '04), June, 2004.

[7] Babcock B., Babu S., Datar M., Motwani R., and Widom J. “Models and issues in data stream systems,” In Phokion G. Kolaitis, editor, Proceedings of the 21nd Symposium on Principles of Database Systems, pages 1-16. ACM Press, 2002.

[8] Geoff Hulten, Pedro Domingos. “Catching up with the data: research issues in mining data streams,” In Proc. of Workshop on Research issues in Data Mining and Knowledge Discovery, 2001.

[9] E.F. Codd et al., “Providing OLAP(On-line Analytical Processing) to User-Analysts: An IT Mandate,” Available from Arborsoft’s Web Site(http://www.arborsoft.com).