DOI: 10.3745/KIPSTD.2009.16-D.4.475

H*-tree/H*-cubing: 데이터 스트림의 OLAP를 위한 향상된 데이터 큐브 구조 및 큐빙 기법

심 상 예[†] · 이 연[††] · 이 동 욱[††] · 김 경 배[†††] · 배 해 영[††††]

요 약

데이터 큐브는 다차원 데이터 분석 및 멀티레벨 데이터 분석에 많이 사용되고 있는 중요한 데이터 구조이다. 최근 데이터 스트림의 온라인 분석에 대한 수요가 증가하면서 스트림 큐브, Flow 큐브, S-큐브 등의 다양한 데이터 큐브 구조와 기법이 제안되었다. 그러나 기존 기법들은 데이터 큐브 생성 시 고비용이 요구되는 단점을 가지고 있어 효과적인 데이터 구조, 질의 방법 및 알고리즘에 대한 연구가 필요하다. 스트림 큐브 기법에서는 H-큐빙 기법을 사용하여 큐보이드를 선택하고, 계산된 셀들을 인기 패스에 있는 큐보이드들로 구성된 H-트리에 저장한다. 그러나 스트림 큐브 기법에서는 H-트리에 데이터를 비순차적으로 삽입하기 때문에 H-큐빙 기법을 사용하여 질의를 처리할 때 제한성을 갖고 있다. 본 논문에서는 데이터의 트리 구조의 각 층에 대한 인덱스를 구축하여 스트림 데이터에 대한 빠른 삽입 연산을 지원하는 H*-tree 구조와, popular-path에 존재하지 않는 큐보이드를 빨리 계산하여 스트림 데이터에 대한 빠른 애드 혹 질의 응답을 지원하는 H*-cubing 기법을 제안한다. 성능평가를 통하여 제안한 H*-tree 기법은 보다 적은 큐브 구축 시간을 지원하며, H*-cubing 기법이 stream cube 기법보다 빠른 애드 혹 질의 응답 시간을 소요하며, 보다 적은메모리를 사용함을 보여준다.

키워드 : OLAP, 데이터 스트림, 데이터 큐브, Ad-Hoc질의 처리

H*-tree/H*-cubing: Improved Data Cube Structure and Cubing Method for OLAP on Data Stream

Xiangrui Chen[†] · Yan Li[††] · Dong-Wook Lee[††] · Gyoung-Bae Kim[†††] · Hae-Young Bae[††††]

ABSTRACT

Data cube plays an important role in multi-dimensional, multi-level data analysis. Meeting on-line analysis requirements of *data stream*, several cube structures have been proposed for *OLAP on data stream, such as stream cube, flowcube, S-cube*. Since it is costly to construct data cube and execute ad-hoc OLAP queries, more research works should be done considering efficient data structure, query method and algorithms. *Stream cube* uses *H-cubing* to compute selected cuboids and store the computed cells in an *H-tree*, which form the cuboids along *popular-path*. However, the *H-tree layoutis disorderly* and *H-cubing* method relies too much on popular path.In this paper, first, we propose H*-tree, an improved data structure, which makes the retrieval operation in tree structure more efficient. Second, we propose an improved cubing method, H*-cubing, with respect to computing the cuboids that cannot be retrieved along popular-path when an ad-hoc OLAP query is executed. H*-tree construction and H*-cubing algorithms are given. Performance study turns out that during the construction step, H*-tree outperforms H-tree with a more desirable trade-off between time and memory usage, and H*-cubing is better adapted to *ad-hoc OLAP queries*with respect to the factors such as time and memory space.

Keywords : On-line Analytic Processing(OLAP), Data Stream, Data Cube, Ad-Hoc Query Answering

1. Introduction

Nowadays, in manufacturing processes, network security, real-time traffic surveillance, and many other kinds of practical areas, data is generated in huge volume, unbounded, time-varying, flowing in-and-out dynamically, and changing rapidly. This kind of potentially infinite data is called *data stream* [1]. Due to the limited resources available in today's computer science technology (e.g. memory, disk, etc.), during the processing of data stream, it may be impossible to store an entire data stream or to

† 준 회 원 : 인하대학교 정보공학과 석사과정
†† 준 회 원 : 인하대학교 정보공학과 박사과정
††† 종신회원 : 서원대학교 컴퓨터교육과 조교수
†††† 종신회원 : 인하대학교 컴퓨터공학부 교수
논문접수 : 2008년 11월 17일
수 정 일 : 1차 2009년 4월 28일
심사완료 : 2009년 5월 20일

scan through it multiple times due to its tremendous volume. Although research on data stream, from storage to indexing, from querying to mining, has been a hot topic fora long time, research with On-line Analytical Processing (OLAP) [2] on data stream is still in its infancy, efficient data structure, cubing method and validated algorithms are needed.

Most of data stream resides at rather low level of abstraction in different application areas,where as analysts are often more interested in higher and multiple levels of abstraction [3]. This new requirement and the success of OLAP technology naturally leads to the new research branch, *OLAP on data stream*, which absolutely differs from the traditional database techniques as they always solely dependon the static and disk-archived data. Obviously, because of the characteristics and restrictions of on-line data stream analysis, mostprevious studies on data stream are not applicable in the field of *stream OLAP*. Reviewing relevant traditional data warehousing and OLAP techniques, *data cube* [4] becomesthe essential component in most data warehouse systems and has been playing an important role in multi-dimensional multi-level data analysis."Is it possible to construct a new cube to facilitate the OLAP on stream?" Answering this question, the framework of *stream cube* is set and proposed as a general data cube structure for *stream OLAP* [3, 5, 7]. Different from previousresearch works on stream query processing and data stream mining, this framework focus more on the multi-dimensional and multi-level on-line analysis of data stream. *Stream cube* is a kind of partially materialized cube, using H-cubing to compute the selected cuboids and H-tree to store the computed cells [6], which form the cuboids along popular-path. On a macro level, to facilitate fast on-line multi-dimensional analysis of data stream, stream cube is composed of three important techniques: (i) tilted time frame - a multi-resolution model to register time-related data which ensures the low memory and disk space usage, (ii) critical layers - a minimal interesting layer (m-layer) and an observation layer (o-layer) which optimize the cubing method according to the interests of analyzer, (iii) novel data stream cubing method based on popular-path which makes a reasonable trade-off between space, computation time and flexibility.

During OLAP, it is alwaysrequired to answer ad-hocaggregate queries over data streams. Considering this challenging task, *prefix aggregate tree (PAT)* is proposed and aimed to facilitate online warehousing [8]. Although it can be regarded as an extension of H-tree to support aggregation queries for data stream, there are still some potential problems should be discussed. First, different from H-tree, it does not take into account the concept hierarchy. This will make the multi-dimensional multi-level OLAP analysis impossible. Second, high level data should be stored in data warehouse for *stream OLAP*. On the contrary, PAT needs to delete records in the tree during dynamical incrementally maintaining with the incoming of data stream without any backup operations. Third, compared with previous proposed approximate methods for aggregate queries, PAT commits to provide accurate answers, which may not always be necessary in reality, considering the semantics of OLAP. Consequently, as an extension of H-tree, it can not be successfully adopted for OLAP on data stream without improvements. Instead, our goal is to provide an improved structure that can be directly used in *stream OLAP*.

Recently, except for *stream cube*, there are severalother cube-structures and data-warehouse has been proposed to support OLAP on stream-like data. For example, the *flowcube* referred in [9] and *RFID warehouse* given in [10], which focus on the multi-dimensional analysis of commodity flows. Another example is the *sequence data cube* (S-cube) and *sequence cuboids* introduced in [11], composed of which, another prototype system named "*Sequence OLAP*" is able to support "pattern-based" grouping and aggregation for OLAP on sequence data. Actually, these two cubes are both devised to facilitate special application cases.

Following the steps of *stream cube* research, we examine and improve this fundamental framework to some extent, with respect to the limitation of response time and memory space. Considering the semantics of OLAP on data stream, first, by running numerous experiments on real datasets, we are able to extend H-tree to H*-tree, which plays the role of in-memory data structure for storing and computing *stream cube*. Unlikethe disorderly layout of H-tree, H*-tree is more regularly as every layer is indexed by a "*binary tree*". Within this indexed structure, the efficiency of tree construction and most query answering can be improved a lot as they both need to retrieve some cells in the tree. In fact, the cause is that the regularly indexeddata structure is always better adapted than disorderly ones. Second, we proposed a novel cubing method, H*-cubing. Reviewing H-cubing, it aims to answer OLAP queries based on the pre-materialized cuboids along popular-path. However, examining some OLAP queries when the inquired cuboids can not be found along popular-path, the

method in H-cubing "to compute use base cuboid at m-layer" is inefficient. To maximally use the materialized cuboids, H*-cubing is expected to compute using the most approximate cuboids that can be found. This strategy shortened the computing time to some extent by utilizing the pre-computed cuboids. Third, relevant algorithms have been given and implemented in this paper. During the performance study, it turns out that H*-tree outperforms H-tree, and H*-cubing is better adapted to answering stream OLAP queries compared with H-cubing, with respect to the factors such as response time and memory space usage.

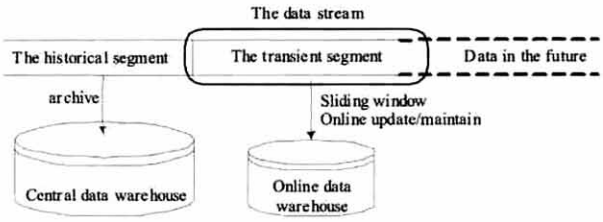
The rest of the paper is organized as follows. In section 2, the potential problems exist in *stream cube* are defined. Section 3 describes H*-tree and gives the construction algorithm. Section 4 is dedicated to present a new algorithm for answering OLAP queries, H*-cubing, focusing on the queries related to some cuboids that can not be found along popular-path. We evaluate the effectiveness of H*-tree and H*-cubing in Section 5 based on the experiments with real datasets. Finally, thepaper is concluded in section 6.

2. Related Work

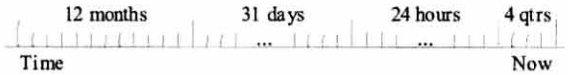
Our work is related to OLAP on data stream, and the most relevant work to ours is *stream cube* [3]. Although novel features of *stream cube*, low construction cost, completeness and compactness well fulfill the challenges of unbounded data stream [3], it still can be improved for more efficient processing. For instance, in *stream cube*, H-cubing is used to compute the selected cuboids and H-tree is adopted to store the computed cells. But it is unclear how H-tree can be stored and incrementally maintained effectively for data streams, and the same question has been put forward in [8]. In this section, we will explain these potential unsolved problems exist in *stream cube* and related concepts.

2.1 OLAP on Data Stream

Reviewing the characteristics of data stream, massive, temporally ordered, varying, and potentially infinite, although research with it has been a hot topic for a long time [1], most of them only focus on the indexing and query answering, not includingthe analysis of data stream or data stream warehouse. Also, traditionalOLAP and data mining methods typically require multiple scans of data and are therefore infeasible for data stream applications. Naturally, the framework of *stream cube* is devised to



(Fig. 1) The framework of warehousing data streams



(Fig. 2) A tilt time frame model

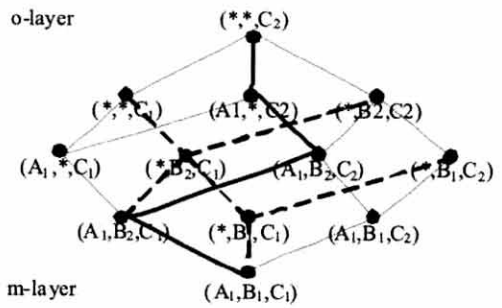
facilitate multi-dimensional multi-level analysis of data stream [3]. Meanwhile, for OLAP, a framework of warehousing data steams is also needed, and in the framework shown in (Fig. 1) [8], a data stream is divided into two segments: the *transient segment* and the *historical segment*. All these works are intended to address the research issues of multi-dimensional OLAP on data stream.

In the design of *stream cube*, within the tilt time frame model, shown in (Fig. 2), one can register time at different levels of granularity.

Also, two critical cuboids (m-layer, o-layer) are identified due to their conceptual and computational importance in stream data analysis. Since *stream cube* is devised as partial materialized structure, the pre-aggregation computation will be along the popular-path by rolling up a computed cuboid residing at the closest lower level. Here, we present these ideas using an example.

[Example 1]

As the *stream cube* shown in (Fig. 3), suppose the stream data to be analyzed contains3 dimensions, A, B, C. Meanwhile, each dimension has 2, 3, 3 levels of



(Fig. 3) Stream cube structure from m-layer to o-layer

abstraction respectively, as (A_1, *), (B_1, B_2, *), (C_1, C_2, *), and * represents the highest level of abstraction. Obviously, it forms a high-to-low hierarchy, where the ordering is "* > A_1", "* > B_2 > B_1", and "* > C_2 > C_1". In this cube, (A_1, B_1, C_1) and (*, *, C_2) are specified as the minimal interesting layer (m-layer) and the observation layer (o-layer). According to the semantics of stream cube, from m-layer (the bottom cuboid) to o-layer (the top-cuboid), there are in total 2×3×2 = 12 cuboids in this example. Suppose the popular-path (drilling path) have been given by user or experts, the darkened path in red shown in (Fig. 3) represents the path along which the drilling-down operation will be executed, (*, *, C_2) → (A_1, *, C_2) → (A_1, B_2, C_2) → (A_1, B_2, C_1) → (A_1, B_1, C_1).

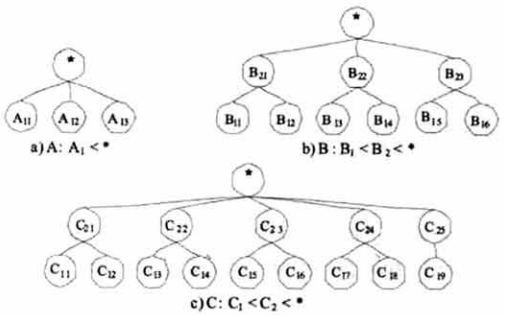
2.2 Disorderly Layout of Layers in H-tree

As referred above, *stream cube* is proposed to support on-line, multi-dimensional analysis of data stream. After examining this structure taking into account the characteristics of data stream carefully,on a macro level, we firmly believe this is a well-devised cube structure for OLAP on data stream. However, on a micro level, there is a potential problem exists in H-tree, *the disorderly layout of every layer*.

According tothe semantics of H-tree, all the concept levels of each dimension involved in the popular-path will be located in different levels of H-tree, and the nodes of each level will store the computed cell values, which then form the corresponding cuboids in *stream cube*. Moreover, the order of the down-to-top cuboids stored in H-tree levels solely depends on the low-to-high order of the cuboids along popular-path. The order of top-to-down layers involved in H-tree only depends on the high-to-low appearance order of the dimensions along popular-path. As to describe the problem clearly, here we first give an H-tree construction example as follows:

[*Example 2*]

Suppose we want to construct an H-tree for the cube shown in (Fig. 3), first, we define the concept hierarchy of A, B and C as shown in (Fig. 4)Obviously, the cardinality of A_1, B_1, C_1, B_2, C_2 is 3, 6, 9, 3, 5 respectively. Reviewing (Fig. 3), (A_1, B_1, C_1), (*, *, C_2) is specified as the m-layerand o-layer. The high-to-low drilling path: (*, *, C_2) → (A_1, *, C_2) → (A_1, B_2, C_2) → (A_1, B_2, C_1) → (A_1, B_1, C_1) is specified as the popular-path. And the top-to-down order in H-tree is: root → C_2 → A_1 → B_2 → C_1 → B_1. Then, the nodes in B_1, C_1, B_2, A_1, C_2 levels will be used to store the cells of (A_1, B_1, C_1), (A_1, B_2, C_1), (A_1, B_2, C_2), (A_1, *, C_2), (*, *, C_2) respectively, which then form the corresponding cuboids.



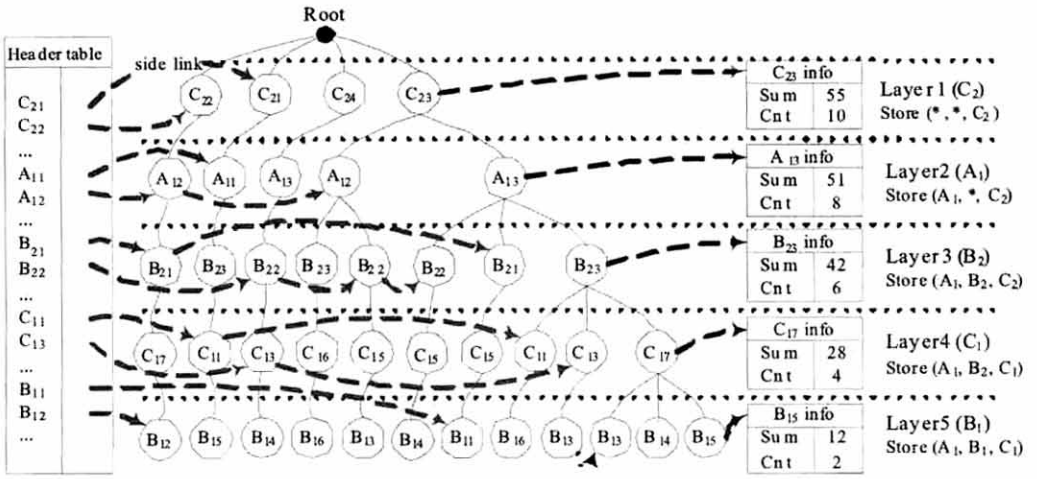
(Fig. 4) Concept hierarchy of the dimensions

What's more, according to the "critical layer" feature of stream cube, the raw data stream will be firstly generalized to the schema of m-layer, (A_{1x}, B_{1y}, C_{1z}). And then, according to the H-tree construction method, the firstly generalized data should secondly be extended to the schema as the top-to-down layer order in H-tree, (C_{2a}, A_{1x}, B_{2b}, C_{1z}, B_{1y}). A base info table for m-layer data example, which holds the values, is shown in <Table 1>.

Let's consider the first tuple t_1(A_{12}, B_{12}, C_{17}, 2) as an example, which isgeneralized to m-layer schema from raw data stream. According to (Fig. 4), the value of B_1 (B_{12}) belongs to the conception of B_2's value (B_{21}), and the value of C_1 (C_{17}) belongs to the conception of C_2's value (C_{24}). As a result, (A_{12}, B_{12}, C_{17}) should be secondly extended to (C_{24}, A_{12}, B_{21}, C_{17}, B_{12}). After the generalization of every tuple has been done, it can be used to construct the H-tree. Given *base info* in <Table 1>, an H-tree can be built shown in (Fig. 5):

<Table 1> Base info table for m-layer data

time	C_2	A_1	B_2	C_1	B_1	measure
t_1	C_{22}	A_{12}	B_{21}	C_{17}	B_{12}	2
t_2	C_{21}	A_{11}	B_{23}	C_{11}	B_{15}	1
t_3	C_{24}	A_{13}	B_{22}	C_{13}	B_{14}	3
t_4	C_{23}	A_{12}	B_{23}	C_{16}	B_{16}	3
t_5	C_{23}	A_{13}	B_{22}	C_{15}	B_{14}	5
t_6	C_{23}	A_{12}	B_{22}	C_{15}	B_{13}	1
t_7	C_{23}	A_{13}	B_{21}	C_{15}	B_{11}	4
t_8	C_{23}	A_{13}	B_{23}	C_{13}	B_{13}	8
t_9	C_{23}	A_{13}	B_{23}	C_{17}	B_{14}	7
t_{10}	C_{23}	A_{13}	B_{23}	C_{17}	B_{15}	5
t_{11}	C_{23}	A_{13}	B_{23}	C_{17}	B_{13}	9
t_{12}	C_{23}	A_{13}	B_{23}	C_{11}	B_{16}	6
t_{13}	C_{23}	A_{13}	B_{23}	C_{17}	B_{15}	7
.....						



(Fig. 5) H-tree structure for stream cube computation (except for the 5 nodes in each layer, the computed info of other nodes are omitted)

From Example 2 presented above, we may find H-tree is well adapted to *stream OLAP* compared with other tree structure since it takes into account the concept hierarchy, which is the essential component of multi-dimensional multi-level analysis in OLAP. However, there is a potential problem should be discussed against this compact tree, *disorderly layout of every layer*. Every time when a new tuple incomes, take t_{13}(C_{23}, A_{13}, B_{23}, C_{17}, B_{15}, 7) for example, it will first retrieve the existed nodes in the first layer (C_2) for the node labeled 'C_{23}'. If it returns 'true', t_{13} will share the found node, or else, it will create a new node labeled C_{23}. Similarly, it will retrieve the existed nodes in the second layer (A_1), searching for the node labeled 'A_{13}' in the children nodes of C_{23}, and the steps will recursive like this in every other layer. On one hand,this dynamic construction method can save memory to some extent, comparing with constructing complete H-tree before insertion. However, on the other hand, efficiency of the retrieval for existed nodes at every so constructed level will be a potential bottleneckfor OLAP, especially with the growing cube size and H-tree levels (the length of popular-path). Understandably, the essential cause of this bottleneck is just the problem, *disorderly layout of every layer*. Then, it comes forth the question, if we want to retrieve a specified node in a layer, which kind of layouts maybe better, disorderly, orderly or with some index? For disorderly layout, mostly 'sequence search' is the best option. For orderly layout, 'binary search' is the best choice. Besides, the layout with "*binary tree*" index also facilitates node retrieval. Considering the third option, we will adopt "*binary tree*" as the indices for every layer of H*-tree in Section 3.

2.3 Ad-hoc Query Answering Efficiency

For *stream cube*, based on H-tree, the algorithm for ad-hoc query answering is called H-cubing, first given in [7] and later revised in [3] for online processing of stream OLAP query. Considering H-cubing [3], it aims to answer OLAP queries based on the pre-materialized cuboids along popular-path. However, examining some OLAP queries when the inquired cuboids can not be found along popular-path, the method in H-cubing "to compute use base cuboid at m-layer" is inefficient.

For a multi-dimensional multi-level stream query, it always specifies a few instantiated constants and inquired information corresponding to one or some dimensions. Consequently, for simplicity, an OLAP query can be considered involving a set of instantiated dimensions, {D_i, ···, D_k}, and a set of inquired dimensions, {D_l, ···, D_p}. The query is then abstracted to a set of relevant dimensions, D_r, which is the union of the sets of instantiated dimensions and the inquired dimensions [3]. For maximal use of the pre-computed cells stored in every layer of H-tree, one needs to find the highest-level popular-path cuboids that contain D_r. If the relevant cuboids are found, the computation can be performed by selecting the relevant data info from the so found cuboids and return results. However, as a matter of fact, sometimes we can not find the relevant cuboids for ad-hoc OLAP queries along popular-path. Although the algorithm described in [3] proposed a solution, "to use the base cuboid at the m-layer to compute it", the pre-computed cells in the lower level have not been maximal usedeither. Meanwhile, taking into account the problem "*disorderly layout of every layer*" defined, when

the popular-path is very long, which means the height of H-tree is considerable, the solution "to use the base cuboid at the m-layer to compute it" will possibly affect the efficiency and ad-hoc query answering quality, increasing the respond time to some extent. Meet this problem and to maximally use the materialized cuboids, we proposed a new cubing method, H*-cubing, whichis expected to compute using the most approximate cuboid that can be found. In section 4, we will give a detailed introduction about it.

Overall, although stream cube has been implemented in the MAIDS project [12], our task is to make it more efficient and fantastic.

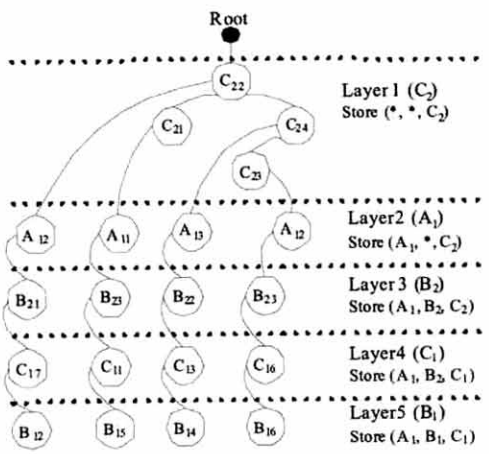
3. Indexing scheme of H*-tree

In this section, we devise the compact data structure, H*-tree. Also, we give the algorithm to construct H*-treeand compare it with relevant works.

3.1 Data Structure

Reviewing Example 2, let the generalized data stream be S (T, C_2, A_1, B_2, C_1, B_1, M), where T and M are the attributes of *time-stamps* and *measure* respectively. The first 13 tuples are shown in <Table 1>, and aggregate function *Sum* and *Cnt* areadopted, by aggregating which we can also get the aggregated value *Avg* (*Sum* / *Cnt*).

In fact, when S arrivesm-layer, the attribute T is to facilitate the storing and aggregation based on titled time frame in every node, and the value of attribute M will be stored and aggregated in every node. Consequently, we can ignore the time-stamps and measures and construct an H*-tree. For instance, tuples t_1, t_2, t_3, t_4, can be organized into an H*-tree shown in (Fig. 6) In order to



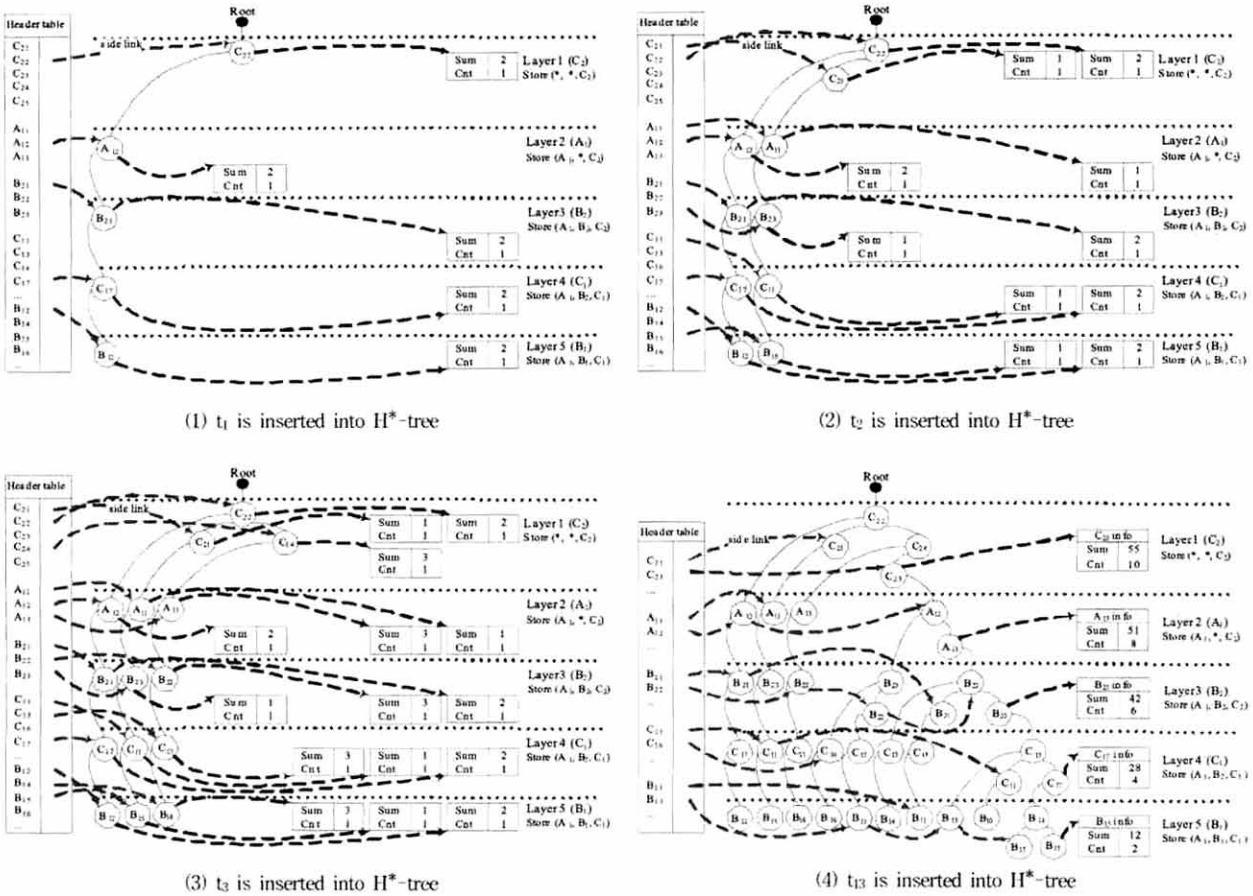
(Fig. 6) H*-tree structure for stream cube computation

store the information about the time-stamps and measures, we can register the raw information at the leaf node and the pre-computed cells in the non-leaf nodes. Each node has an *aggregated info table*, so that the time-stamps and the aggregates by tuples can be registered.

3.2 H*-tree Construction Steps

For simplicity, given *base info* in <Table 1>, an H*-tree can be built following steps, briefly illustrated in (Fig. 7):

1. H*-tree has a root node "null", and the top-to-down order of dimensions is also depend on the appearance order along popular-path, root → C_2 → A_1 → B_2 → C_1 → B_1.
2. A *header table* is created, in which each entry represents the attribute-value of every dimension appears in the specified popular-path. This is used to link nodes that have the same label in every layer of H*-tree.
3. The first tuple, t_1 = (C_{22}, A_{12}, B_{21}, C_{17}, B_{12}, 2), is inserted into H-tree, with five nodes, C_{22}, A_{12}, B_{21}, C_{17}, and B_{12} inserted in sequence to form the first branch, shown in (Fig. 7) a), and the measure info in the leaf (B_{12}). Also, they are linked to head table by the side links. Suppose the value user interested is the average measure, as to pre-compute the other cuboids stored in the upper levels, the m-layer value (sum = 2, cnt = 1) should be aggregated to the higher levels in the tree, and the aggregate method is to add up the values of all children nodes, then store the result in parent node in the higher level. Following this method, info stored in the nodes along the first branch is the same (sum = 2, cnt = 1).
4. Similarly, t_2, t_3, are insertedand form the other 3 branches, with the higher level node info updated. However, as described above and shown in (Fig. 7) b), c), all the nodes in every layer of H*-tree will be indexed following the "*binary tree*" structure. For instance, when t_2 is inserted, it first retrieves the existed nodes in every layer for the same labeled nodes from top to down. Since C_{21} differs from the existed C_{22} created before, it will create another node in the first layer as the left child of C_{22}, with other nodes followed and form the second branch. Similarly, as C_{24} is larger than C_{22}, t_3 creates a right child node of C_{22}.
5. The remaining tuples, t_4 ⋯ t_{13}, is inserted one by one following the same steps, with the higher level node info updated. Since t_{13} and t_{10} have the same attribute values, they can share the single path, with node info in the leaf (sum = 12, cnt = 2). As a limited of space, in (Fig. 7) d), we only listthe aggregated info of five nodes,



(1) t_1 is inserted into H*-tree

(2) t_2 is inserted into H*-tree

(3) t_3 is inserted into H*-tree

(4) t_{13} is inserted into H*-tree

(Fig. 7) H*-tree construction steps. In (4), except for the 5 nodes in each layer, the computed info of other nodes are omitted

others are omitted. Predictably, the "*binary tree*" in every layer of H*-treewill be of great worth for the efficiency of construction and H*-cubing.

6. Finally, the tree so formed in (Fig. 7) d) is called an H*-tree. Its construction requires only one scan of data stream, and makes the nodes in the tree well organized,which meets the feature of data stream and be well adapted to *stream OLAP*.

Here, as a supplementary, we give the construction algorithmof H*-tree in (Fig. 8)

3.3 Comparison: H*-tree Versus Previous Methods

Binary tree structure has been widely approved as a solution for indexing records in large volume of datasets. H*-tree is another tree involving this classicalstructure. Meanwhile, there has been a lot of tree structures been proposed. However, there are some essential differences. Some detailed differences withprevious researches have been given in [8], here we just compare with H-tree and PAT in the following two aspects.

First, Compared with H-tree structure shown in Fig. 5,

Algorithm 1: Constructing an H*-tree
Input: Pre-processed data stream S
Output: An H*-tree for storing and computing of stream cube
Method:
1. Begin
2. S = (*,...,*);
3. Generalize S into the schema S' involving dimensions appeared along popular-path from high to low;
4. Create a HeadTable include all attributes of all appeared dimensions along popular-path;
5. Create a node as the root of H*-tree;
6. H*_LayerCnt = the depth of H*-tree,
Cur_H*_Layer = Currently inserted layer;
//Insert tuples in S' one by one into H*-tree
7. if (Cur_H*_Layer == H*_LayerCnt) //the leaf node
Retrieve the existed nodes to find the same labeled one;
if Success
Update the found node info and the the upper nodes in H*-tree;
else
Create a left or right child node of the parent node and store info;
Aggregate the upper nodes of layers in H*_tree;
return;
8. else //the nonleaf node
Retrieve the existed nodes to find the same labeled one;
if Success
Set the created node as parent;
else
Create a left or right child node of the parent node;
Set the created node as parent;
Recall H*-tree constructing method;
9. End

(Fig. 8) H*-tree construction algorithm

the obvious difference between these two trees is the layout. Meet the problem of H-tree, *disorderly layout of every layer*, we index the tree structure in every layer following "*binary tree*" method. Clearly, layers with the *binary tree* indices will bewell adapted not only to the retrieval of existed nodes in every layer when a new tuple incomes, but also to the ad-hoc OLAP query answering which sometimes need to select and aggregate cells stored in different levels of H*-tree. Essentially, this kind of structure makes H*-tree solve the potential problem, *disorderly layout of every layer*, and makes the tree structure well organized for stream OLAP.

Second, PAT, the same as H*-tree, can be regarded as an extension of H-tree. It is indexed by infix links and side links. Compared with PAT, H*-tree is more compact and better organized by *binary tree* index. What's more, PAT focus more on the efficiency of answering ad-hoc aggregate queries from data stream, but ignores the essential of OLAP, multi-dimensional multi-conceptual analysis.Consequently, it may need more improvements to facilitate stream OLAP ad-hoc queries. However, the infix link adopted in PAT is interesting, and it may lead a new angle of view for the devising of tree.

4. Ad-hoc Query Answering

In this section, we will focus on the discussion about how to answer ad-hoc OLAP queries utilizing H*-tree maximally by proposing H*-cubing method.

4.1 H*-cubing Method

Mostly, an OLAP query can be considered involving a set of instantiated dimensions, $\{D_i, \cdots, D_k\}$, and a set of inquired dimensions, $\{D_l, \cdots, D_p\}$. The query is then abstracted to a set of relevant dimensions, D_r which is the union of the sets ofinstantiated dimensions and the inquired dimensions. In every layer of H*-tree and H-tree, the existed nodes can be taken as the collectionof partial-materialized stream cells, which form the cuboids along popular-path. Defined in H-cubing, to find the highest-level popular-path cuboids that contain D_r is to maximally utilize the materialized popular-path. And for any query, which corresponds to some cells in some cuboids, if the pre-aggregated cuboids containing D_r can be found, the result can be given by some "select" operation directly. However, the solution in H-cubing for computing D_r in occasion that it can not be found in the pre-aggregated cuboids along popular-path, is to use the base cuboid at the m-layer to compute. This does not maximally utilize the pre-computed cells in the lower level. From that point of view, H*-cubing will be a more desirable approach as it will maximally utilize H*-tree by retrieving the most approximate cuboids instead of using the base cuboid directly.

4.2 Query answering using H*-cubing

As an extension of H-cubing, our task is to maximally utilize the materialized H*-tree. For simplicity, we will describe the query answering method, H*-cubing, based on the H*-tree created shown in (Fig. 7) d).

First, consider query Q_1: (A_{13}, B_{23}, C_{23}), which itself is an aggregated cell, stored in the node belonging to the 3rd layer of the constructed H*-tree. Its aggregate value, 42 (sum), 6 (cnt), is registered in the aggregate table of node $C_{23}A_{13}B_{23}$. Following the path from the root to the node, we can retrieve the answer quickly. The query answering steps is as follows:

1. Q_1 is abstracted to instantiated dimensions, {A_1, B_2, C_2} and no inquired dimensions. Then, D_r = {A_1, B_2, C_2}.
2. Retrieve the cuboids along popular-path from top to down, (*, *, C_2) → (A_1, *, C_2) → (A_1, B_2, C_2) → (A_1, B_2, C_1) → (A_1, B_1, C_1), to see whether there is an cuboid which covers D_r. Obviously, the third cuboid (A_1, B_2, C_2) stored in the 3rd layer of H*-tree is the objected cuboid.
3. Combined with Q_1, we need to do the "select" operation upon the so found cuboid to return the aggregated cell in it.

Sometimes, in the third step, to return the final result, side link sometimes should be used to aggregate the same labeled nodes, stored in the same layer of H*-tree. However, the method given above only considers the cuboids being found condition. Next, we will give another query answering method focus on the condition that the objected cuboid can not be found.

Let us consider query Q_2: (*, B_{23}, C_{23}), and the query answering steps is as follows:

1. Q_2 is abstracted to instantiated dimensions, {*, B_2, C_2} and no inquired dimensions. Then, D_r = {*, B_2, C_2}.
2. Retrieve the cuboids along popular-path from top to down, (*, *, C_2) → (A_1, *, C_2) → (A_1, B_2, C_2) → (A_1, B_2, C_1) → (A_1, B_1, C_1), to see whether there is an cuboid which covers D_r. Obviously, none cuboid includes the three items, which means the cuboid contain the query result is not an pre-aggregated one along popular-path. In H-cubing, it is suggestedto use the base cuboid (A_1,

Algorithm 2: Query based on H*-tree
Input: H*-tree, Query Q
Output: Query result
Method:
1. Begin
2. Relevant dimensions of Q = D_r;
3. Retrieve the cuboid involving D_r exists on popular-path from top to down;
4. if Success
   Do "select" operation on the so found cuboid;
   Return result;
5. else
   Retrieve the most approximate cuboid involving D_r exists on popular-path from top to down;
   Aggregate condition check upon the so found cuboid, true or false;
6. if true
   Do aggregate operation based on the so found cuboid;
   Return result;
7. else
   Do aggregate operation based on the m-layer cuboid;
   Return result;
8. End

(Fig. 9) Query answering using H*-cubing

B_1, C_1) to compute (*, B_{23}, C_{23}) immediately. Nevertheless, H*-cubing will first select the most approximate pre-aggregated one from top to down along popular-path. Here, the first most approximate one is (*, *, C_2), however, as * is higher than B_2 in the concept hierarchy, we can not aggregate from higher levels to lower ones. Then, the next approximate one is (A_1, B_2, C_2), and A_1 is lower than *, which means it is right the most approximate cuboid we need.

3. Combined with Q_2, we need to do some aggregate operations to return (*, B_{23}, C_{23}). Based on the found cuboid, (A_1, B_2, C_2), aggregate all the cells contain B_{23} and C_{23} upon first dimension. Then, we can return the value of (*, B_{23}, C_{23}).

Summarized from the query answering steps of Q_1 and Q_2, H*-cubing algorithm can be given in (Fig. 9).

4.3 Comparison: H*-cubing Versus H-cubing

Compared with H-cubing, we more maximally utilized the constructed H*-tree. Essentially, considering the condition that the cuboids involving D_r can not be found along popular-path, instead of computing using the base cuboid directly in H-cubing, H*-cubing first recursively retrieve the most approximate cuboids in H*-tree. These sofound cuboids have been aggregated and stored in H*-tree when it is constructed. Regarding computation, aggregation based on m-layergiven by H-tree will lead to more computation, and it may not utilize the intermediate approximate cuboid as effectively as H*-cubing. As a proof, a performance study is needed to show the efficiency of H*-cubing.

5. Performance Evaluation

In this section, to evaluate the efficiency of H*-tree and H*-cubing, we performed an extensive performance analysis on real datasets. All experiments were conduced on a 2.6 GHz Pentium 4 PC with 2.0GB main memory, running Microsoft-XP Professional. All methods are implemented using Microsoft Visual C++ 6.0. We compared the performance of H*-tree with H-tree, H*-cubing with H-cubing. Since the original H-tree and H-cubing code is unavailable, we implement it as efficiently as possible.

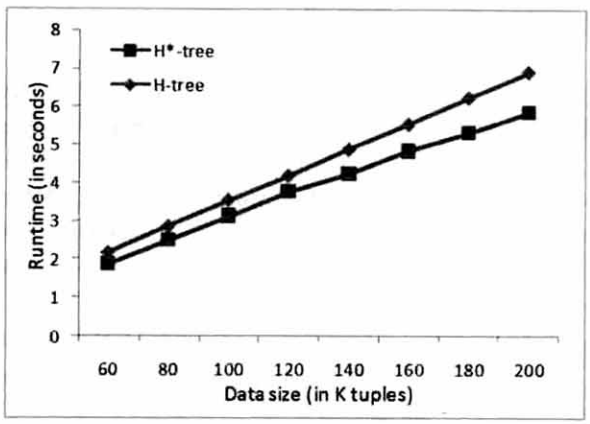
We have examined the following factors in our performance study. For the construction efficiency of H-tree and H*-tree, (1) time and space with respect to the data size; (2) time and space with respect to the popular-pathlength (the number of cuboids along popular-path). For the evaluation of ad-hocquery answering efficiency, time and sp ace with respect to the data size is considered.

5.1 About the Datasets

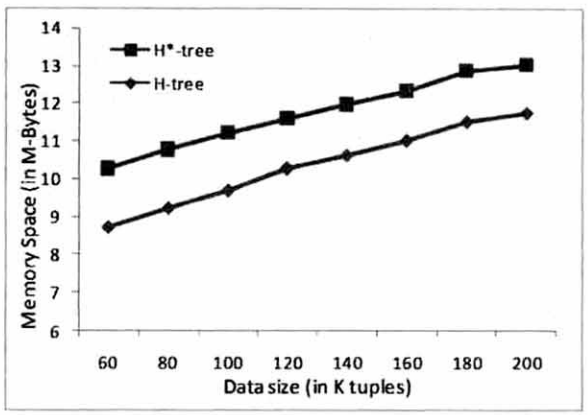
Here we report our performance study with real datasets. This real datasets is about traffic information such as flow, speed, and density, which is obtained from actual detailed vehicle trajectories collected by Cambridge Systematics,Inc., under the auspices of the Next Generation SIMulation (NGSIM) program [13]. The NGSIM program has collected data sets of vehicle trajectories from actual live traffic video footages. To the best of our knowledge, traffic field data with such a level of detail were previously unavailable. Attributes extracted from the original datasets in this paper are under the schema *flow(Global Time, Vehicle Class, Lane Identification, Local X, Local Y, Vehicle Velocity, Vehicle Acceleration)*, where *Global Time* is the time-stamp, *Vehicle Class*, *Lane Identification*, *Local X* and *Local Y* are the dimensions, *Vehicle Velocity* and *Vehicle Acceleration* are the measures. You may find more detailed description about it in [13].

5.2 Experiment 1: Construction of Trees

(Fig. 10) shows the processing time and memory usage for the construction of H-tree and H*-tree, with increasing size of the datasets, where the size is measured as the number of tuples at m-layer for the datasets, the popular -path length (the number of cuboids along popular-path) is 9, which means the height of the trees is 9. Since the layout of H*-tree is indexed by "*binary tree*", the

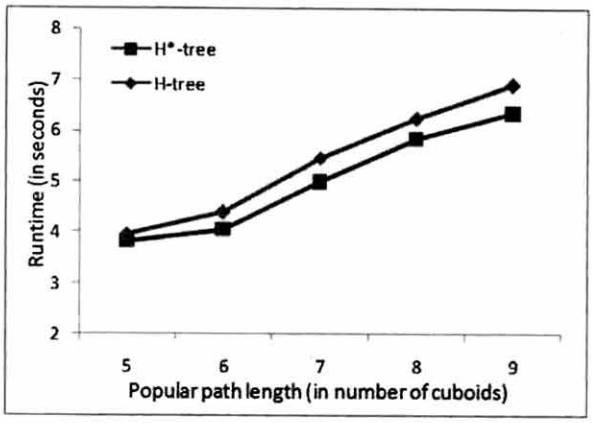


(1) Time vs. size

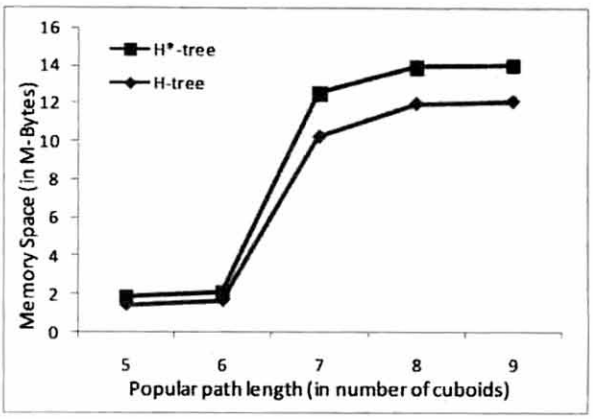


(2) Space vs. size

(Fig. 10) Cube computation: time and memory usage vs. no. tuples at m-layer



(1) Time vs. length



(2) Space vs. length

(Fig. 11) Cube computation: time and memory usage vs. popular-path length

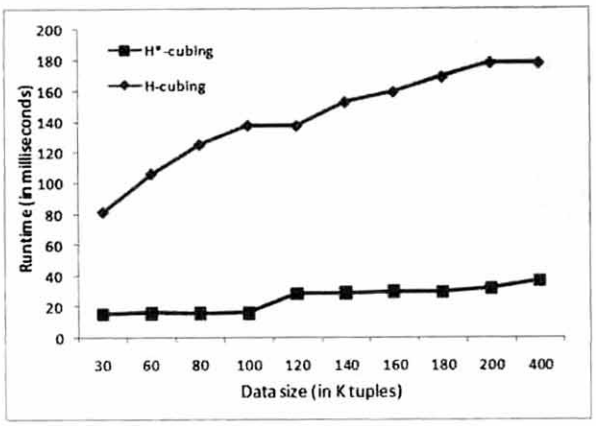
response time is shorter than H-tree shown in (1). With the growing of data size, the trend will be more obvious. However, shown in (2), as the construction of H*-tree is more complex than H-tree, the memory usage isa little higher.

(Fig. 11) shows the processing time and memory usage for the construction of H-tree and H*-tree, with increasing length of popular-path, which means the height of the trees. As been discussed before, the length is measured as the number of cuboids along popular-path. Meanwhile, the size of datasets imcoming m-layer is 200K tuples. Since the regularly layout of H*-tree, the response time is shorter than H-tree, shown in (1). On the contrary, the memory usage of H*-tree is higher shown in (2), but stable. Also, there is a "jump" from 6 to 7 in (2). The reason is that the number of tuple dimensions is 3 (ignoring "*Local X*") when the length is under 6, and the number of tuple dimensions is 4 when the length is above 7.From the "jump", we can also conclude that the change of tuple dimensions number has a considerable impact on the performance.

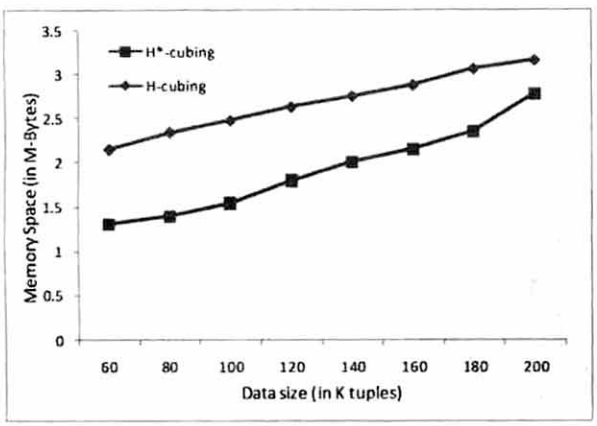
Anyway, considering the features of OLAP on data stream, a more desirable trade-off between response time and memory space should be considered. And this experiment proves that H*-tree is a more ideal option.

5.3 Experiment 2: Ad-hoc Query Answering

(Fig. 12) shows the processing time and memory usage for query answering using H-cubing and H*-cubing, with increasing size of the datasets, where the size is also measured as the number of tuples at m-layer for the datasets, the popular-path length (the number of cuboids along popular-path) is 9. Obviously, due to the "*binary tree*" index of H*-tree and the less computation by utilizing the most approximate cuboids, H*-cubing outperforms H-cubing with respect to both response time and memory usage. Moreover, with the growing of data size, the advantage of H*-cubing will be more obvious.



(1) Time vs. size



(2) Space vs. size

(Fig. 12) Query answering: time and memory usage vs. no. tuples at m-layer

6. Conclusion

On-line analytical processing (OLAP) on data stream is an interesting and challenging research topic. In this paper, we have done some research on the fundamental data structure and algorithms of this target, and succeed in optimizing *stream cube*, based on which more complex OLAP applications can be implemented efficiently. As a proof, we present a performance study withreal traffic datasets to examine the effectiveness and the efficiency of our optimization.

7. Acknowledgement

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.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani and J. Widom, "Models and Issues in Data Stream Systems," Proc. ACM Symposium on Principles of Database Systems (PODS), Madison, Wisconsin, USA, pp.1-16, 2002.

[2] E.F. Codd et al., "Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate," Available: http://www.arborsoft.com

[3] 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 Journal, Vol.18, No.2, pp.173-197, 2005.

[4] 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 Journal, Vol.1, No.1, pp.29-53, 1997.

[5] Y. Chen, G. Dong, J. Han, J. Pei, B. W. Wahand J. Wang, "Online Analytical Processing Data stream: Is It Feasible?" ACM SIGMOD InternationalWorkshop on Research Issues on Data Mining and Knowledge Discovery (DMKD), Madison, Wisconsin, USA, 2002.

[6] J. Han, J. Pei, G. Dong and K. Wang, "Efficient computation of iceberg cubes with complex measures," Proc. ACM SIGMOD International Conference on Management of Data, Santa Barbara, California, USA, pp.1-12, 2001.

[7] Y. Chen, G. Dong, J. Han, B. W. Wah and J. Wang, "Multi-Dimensional Regression Analysis of Time-Series Data Streams," Proc. ACM VLDB International Conference on Very Large Data Bases, Hong Kong, China, pp.323-334, 2002.

[8] M. Cho, J. Pei, and K. Wang, "Answering Ad-hoc Aggregate Queries from Data Streams Using Prefix Aggregate Trees," Knowledge and Information Systems Journal, Vol.12, No.3, pp.301-329, 2007.

[9] H. Gonzalez, J. Han and X. Li, "FlowCube: Constructing RFID FlowCubes for Multi-Dimensional Analysis of Commodity Flows," Proc. ACM VLDB International Conference on Very Large Data Bases, Seoul, Korea, pp.834-845, 2006.

[10] H. Gonzalez, J. Han, X. Li and D. Klabjan, "Warehousing and Analysis of Massive RFID Data Sets," Proc. IEEE ICDE International Conference onData Engineering, Atlanta, Georgia, USA, pp.83 ,2006.

[11] E. Lo, B. Kao, S. Lee, W. Ho, C. Chui and D. Cheung, "OLAP on Sequence Data," Proc. ACM SIGMOD International Conference on Management of Data, Vancouver, Canada, pp.649-660, 2008.

[12] Y. Cai, D. Clutter, G. Pape, J. Han, M. Welge and L. Auvil, "MAIDS: Mining Alarming Incidents from Data Streams," Proc. ACM SIGMOD International Conference on Management

of Data, Paris, France, pp.919-920, 2004.
[13] Cambridge Systematics Inc., "NGSIM (Next Generation SIMulation)," Oakland, California, USA, June, 2005. Available: http://ngsim.camsys.com/.

심 상 예

e-mail : airchenDB@gmail.com
2008년 중국 중경우전대학교 컴퓨터과학 기술학과(공학사)
2008년~현 재 인하대학교 정보공학과 석사과정
관심분야 : 데이터 스트림, OLAP 연산, 유비쿼터스GIS, cloud컴퓨팅

이 연

e-mail : leeyeon@dblab.inha.ac.kr
2006년 중국 중경우전대학교 지리정보 시스템학과(이학사)
2008년 인하대학교 컴퓨터정보공학과 (공학석사)
2008년~현 재 인하대학교 정보공학과 박사과정
관심분야 : 공간 데이터베이스, 공간 데이터웨어하우스, 지리정보 시스템, USN, 스트림 데이터 시스템

이 동 욱

e-mail : dwlee@dblab.inha.ac.kr
2003년 상지대학교 전자계산공학과(이학사)
2005년 인하대학교 컴퓨터정보공학과 (공학석사)
2005년~현 재 인하대학교 정보공학과 박사과정
관심분야 : 유비쿼터스 환경을 위한 공간 DBMS 및 DSMS, 공간 Data Warehouse

김 경 배

e-mail : gbkim@seowon.ac.kr
1992년 인하대학교 전자계산공학과(공학사)
1994년 인하대학교 전자계산공학과(공학석사)
2000년 인하대학교 전자계산공학과(공학박사)
2000년~2004년 한국전자통신 선임연구원
2004년~현 재 서원대학교 컴퓨터교육과 조교수
관심분야 : 이동실시간 데이터베이스, 스토리지 시스템, GIS, VOD

배 해 영

e-mail : hybae@inha.ac.kr
1974년 인하대학교 응용물리학과(공학사)
1978년 연세대학교 전자계산학과(공학석사)
1989년 숭실대학교 전자계산학과(공학박사)
2004년~2006년 인하대학교 정보통신대학원 원장
2006년~2009년 인하대학교 대학원 원장
1982년~현 재 인하대학교 컴퓨터공학부 교수
관심분야 : 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스 등