Online Analytical Processing Stream Data: Is It Feasible?*

Yixin Chen[1], Guozhu Dong[2], Jiawei Han[1,3], Jian Pei[3], Benjamin W. Wah[1], Jianyong Wang[1,4]

[1] University of Illinois at Urbana-Champaign, U.S.A.
[2] Wright State University, U.S.A.
[3] Simon Fraser University, Canada
[4] Beijing University, China

ABSTRACT

Real-time surveillance systems and other dynamic environments often generate tremendous (potentially infinite) volume of stream data: the volume is too huge to be scanned multiple times. However, much of such data resides at rather low level of abstraction, whereas most analysts are interested in dynamic changes (such as trends and outliers) at relatively high levels of abstraction. To discover such high level characteristics, one may need to perform on-line multi-level analysis of stream data, similar to OLAP (on-line analytical processing) of relational or data warehouse data.

With limited storage space and the demand for fast response, is it realistic to promote on-line, multi-dimensional analysis and mining of stream data to alert people about dramatic changes of situations at multiple-levels of abstraction?

In this paper, we present an architecture, called stream_cube, which, based on our analysis, is feasible for successful on-line, multi-dimensional, multi-level analysis of stream data. By *successful*, we mean that the system will provide analytical power and flexibility, derive timely and quality responses, and consume limited memory space and other resources.

The general design of the *stream_cube* architecture is described as follows. First, a **tilt time frame** model is taken as the default model for time dimension. Such a model reduces the amount of data to be retained in memory or stored on disks but still achieves flexibility and analysis power. Second, a small number of **critical layers** are maintained for flexible analysis. Consider that the stream data resides at the primitive layer. It is desirable to identify two critical higher layers in applications: *the minimal interest layer*, and *the observation layer*. These two layers can be used as the basis for stream_cube construction and for multi-level on-line analysis. Third, instead of computing a complete data cube based on the notion of critical layers, one may compute only the layers along a **popular path** and leave others for query-driven, on-line computation. Our analysis shows that this architecture provides a good time-space trade-off: it decreases substantially the demand for memory usage and reduces the overall response time. Other possible designs for OLAPing stream data are also discussed in the paper.

* Work supported in part by grants from NSERC and NCE of Canada, the University of Illinois, and Microsoft Research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2002 ACM DMKD'02:6/03/02 ...$5.00.

1. INTRODUCTION

With years of research and development of data warehouse and OLAP technology [8, 5], a large number of data warehouses and data cubes have been successfully constructed and deployed in applications, and data cube has become an essential component in most data warehouse systems and in some extended relational database systems and has been playing an increasingly important role in data analysis and intelligent decision support.

The data warehouse and OLAP technology is based on the integration and consolidation of data in multi-dimensional space to facilitate powerful and fast on-line data analysis. Data are aggregated either completely or partially in multiple dimensions and multiple levels, and are stored in the form of either relations or multi-dimensional arrays [1, 14]. The dimensions in a data cube are of categorical data, such as products, region, time, etc., and the measures are numerical data, representing various kinds of aggregates, such as *sum*, *average*, and *variance* of sales or profits, etc.

The success of OLAP technology naturally leads to its possible extension from the analysis of static, pre-integrated, historical data to that of current, dynamically changing data streams, including scientific and engineering data, time-series data, and data produced in other dynamic environments, such as power supply, network traffic, stock exchange, telecommunication data flow, Web click streams, weather or environment monitoring, etc.

A fundamental difference in the analysis of stream data from that of relational and warehouse data is that the stream data is generated in huge volume, flowing in-and-out dynamically, and changing rapidly. Due to limited memory or disk space and processing power to handle such huge volumes of data, most data streams may only be examined in a single pass. These characteristics of stream data have been emphasized and investigated by many researchers [3, 7, 6, 9], and efficient stream data querying, clustering and classification algorithms have been proposed recently (such as [13,

10, 7, 6, 9]). However, there is another important characteristic of stream data that has not drawn enough attention: *Most of stream data resides at rather low level of abstraction, whereas an analyst is often more interested in higher and multiple levels of abstraction.* Similar to OLAP analysis of static data, multi-level, multi-dimensional on-line analysis should be performed on stream data as well.

The requirement for multi-level, multi-dimensional on-line analysis of stream data, though desirable, raises a challenging research issue: "*Is it feasible to perform OLAP analysis on huge volumes of stream data since a data cube is usually much bigger than the original data set, and its construction may take multiple database scans?*"

In this paper, we examine this issue and present an interesting architecture for on-line analytical analysis of stream data. Before presenting our architecture, let's first examine an example.

Example 1. Suppose that a Web server, such as Yahoo.com, receives a huge volume of Web click streams, requesting various kinds of services and information. Usually, such stream data resides at rather low level, consisting of time (down to subseconds), Web page address (down to concrete URL), user ip address (down to detailed machine IP address), etc. However, an analyst may often be interested in changes, trends, and unusual patterns, happening in the data streams, at certain high levels of abstraction. For example, it is interesting to find that *the Web clicking traffic in North America on sports in the last 15 minutes is 40% higher than the last 24 hours' average.*

From the point of view of a Web analysis provider, given a large volume of fast changing Web click streams, and with limited resource and computational power, it is only realistic to analyze the changes of Web usage at certain high levels, discover unusual situations, and drill down to some more detailed levels for in-depth analysis, when needed, in order to make timely responses.

Interestingly, both the analyst and analysis provider share a similar view on such stream data analysis: instead of bogging down to every detail of data stream, a demanding request is to provide on-line analysis of changes, trends and other patterns at high levels of abstraction, with low cost and fast response time. ■

In this study, we take Example 1 as a typical scenario and study how to perform efficient and effective multi-dimensional analysis of stream data, with the following contributions.

1. For on-line stream data analysis, both space and time are critical. In order to avoid imposing unrealistic demand on space and time, instead of computing a fully materialized cube, we suggest to compute a partially materialized data cube, with a *tilt time frame* as its time dimension model. In the **tilt time frame**, time is registered at different levels of granularity. The most recent time is registered at the finest granularity; the more distant time is registered at coarser granularity; the level of coarseness depends on the application requirements and on how old the time point is. This model is sufficient for most analysis tasks, and at the same time it also ensures that the total amount of data to retain in memory or to be stored on disk is small.

2. Due to limited memory space in stream data analysis, it is often too costly to store a precomputed cube, even with the *tilt time frame*. We propose to compute and store only two **critical layers** (which are essentially cuboids) in the cube: (1) an *observation layer*, called *o-layer*, which is the layer that an analyst would like to check and make decisions for either signaling the exceptions or drilling on the exception cells down to lower layers to find their corresponding lower level exceptions; and (2) the *minimal interesting layer*, called *m-layer*, which is the minimal layer that an analyst would like to examine, since it is often neither cost-effective nor practically interesting to examine the minute detail of stream data. For example, in Example 1, we assume the *o*-layer is *user-region*, *theme*, and *quarter*, while the *m*-layer is *user*, *sub-theme*, and *minute*.

3. Storing a cube at only two critical layers leaves a lot of room at what to compute and how to compute for the cuboids between the two layers. We propose one method, called popular-path **cubing**, which rolls up the cuboids from the *m*-layer to the *o*-layer, by following one popular drilling path, materializes only the layers along the path, and leave other layers to be computed only when needed. We show that this method represents a reasonable trade-off between space, computation time, and flexibility, and has both quick aggregation time and exception detection time.

The rest of the paper is organized as follows. In Section 2, we present an architectural design for on-line analysis of stream data by defining the problem and introducing the concepts of *tilt time frame* and *critical layers*. In Section 3, we present a popular-path cubing method. A feasibility analysis of the proposed approach is conducted in Section 4. Related work and possible extensions of the method are discussed in Section 5. Our study is concluded in Section 6.

2. ARCHITECTURE FOR ON-LINE ANALYSIS OF DATA STREAMS

In this section, we first introduce the research problem and then present an architecture design for on-line analysis of data streams.

2.1 Problem Definition

In this study, we consider stream data as huge volume, infinite flow of data records, such as Web click streams, telephone calls, on-line transactions, etc. The data is collected at the most detailed level in a multi-dimensional space, which may represent time, location, user, theme, and other semantic information.

To perform multi-dimensional, multi-level analysis, we need to introduce the basic terms related to data cubes.

Let $\mathcal{D}$ be a relational table, called the base table, of a given cube. The set of all attributes $\mathcal{A}$ in $\mathcal{D}$ are partitioned into two subsets, the dimensional attributes DIM and the measure attributes M (so $DIM \cup M = \mathcal{A}$ and $DIM \cap M = \emptyset$). The measure attributes functionally depend on the dimensional attributes in $\mathcal{D}$ and are defined in the context of data cube using some typical aggregate functions, such as COUNT, SUM, AVG, or some regression related measures to be studied here.

A tuple with schema $\mathcal{A}$ in a multi-dimensional space (i.e., in the context of data cube) is called a **cell**. Given three distinct cells c_1, c_2 and c_3, c_1 is an **ancestor** of c_2, and c_2 a

descendant of c_1 iff on every dimensional attribute, either c_1 and c_2 share the same value, or c_1's value is a generalized value of c_2's in the dimension's concept hierarchy. c_2 is a **sibling** of c_3 iff c_2 and c_3 have identical values in all dimensions except one dimension A where $c_2[A]$ and $c_3[A]$ have the same parent in the dimension's domain hierarchy. A cell which has k non-* values is called a **k-d cell.** (We use "*" to indicate "all", i.e., the highest level on any dimension.)

A tuple $c \in \mathcal{D}$ is called a **base cell.** A base cell does not have any descendant. A cell c is an **aggregated cell** iff it is an ancestor of some base cell. For each aggregated cell c, its values on the measure attributes are derived from the complete set of descendant base cells of c.

Our task is to perform high-level, on-line, multi-dimensional analysis of data streams in order to find unusual (exceptional) changes of trends, according to users' interest. This may involve construction of a data cube, if feasible, to facilitate on-line, flexible analysis.

2.2 A stream_cube architecture

To facilitate on-line, multi-dimensional analysis of data streams, we propose a stream_cube architecture with the following features: (1) *tilt time frame*, (2) two *critical layers*: a *minimal interesting layer* and an *observation layer*, and (3) *partial computation of data cubes*. The stream data cubes so constructed are much smaller than those constructed from the raw stream data but will still be effective for usual multi-dimensional stream data analysis tasks.

2.2.1 Tilt time frame

In stream data analysis, people are usually interested in recent changes at a fine scale, but long term changes at a coarse scale. Naturally, one can register time at different levels of granularity. The most recent time is registered at the finest granularity; the more distant time is registered at coarser granularity; and the level of coarseness depends on the application requirements and on how old the time point is (from the current time).

Figure 1: A tilt time frame model

Example 2. For Ex. 1, a *tilt time frame* can be constructed as shown in Figure 1, where the time frame is structured in multiple granularities: the most recent 15 minutes, then the last 4 quarters, 24 hours, and 7 days. Based on this model, one can compute measures in the last 15 minutes with the precision of minute, the last one hour with the precision of quarter, the last day with the precision of hour, and so on, until the whole week with the precision of day. This model registers only $15 + 4 + 24 + 7 = 50$ units of time instead of $7 \times 24 \times 60 = 10{,}080$ units, a saving of about 200 times. Notice that with this model, it is easy to answer queries like "*comparing the average Web traffic (or any measure registered) in the last minute against that in the last hour, last 24 hours, and last 7 days.* However, it is difficult to compare the average traffic in this minute with that of any particular minute in the last seven days. Nevertheless, it is an acceptable trade-off between efficiency and the grain of granularity at a distant time. ■

This model is sufficient for usual time-related queries, and at the same time it ensures that the total amount of data to retain in memory and/or be computed is small.

2.2.2 Critical layers

Even with the *tilt time frame* model, it could still be too costly to dynamically compute and store a full cube since such a cube may have quite a few dimensions, each containing multiple levels with many distinct values. Since stream data analysis has only limited memory space but requires fast response time, a realistic arrangement is to compute and store only some mission-critical cuboids in the cube.

In our design, two critical cuboids are identified due to their conceptual and computational importance in stream data analysis. We call these cuboids layers and suggest to compute and store them dynamically. The first layer, called *m-layer*, is the minimally interesting layer that an analyst would like to study. It is necessary to have such a layer since it is often neither cost-effective nor practically interesting to examine the minute detail of stream data. The second layer, called *o-layer*, is the observation layer at which an analyst (or an automated system) would like to check and make decisions of either signaling the exceptions, or drilling on the exception cells down to lower layers to find their lower-level exceptional descendants.

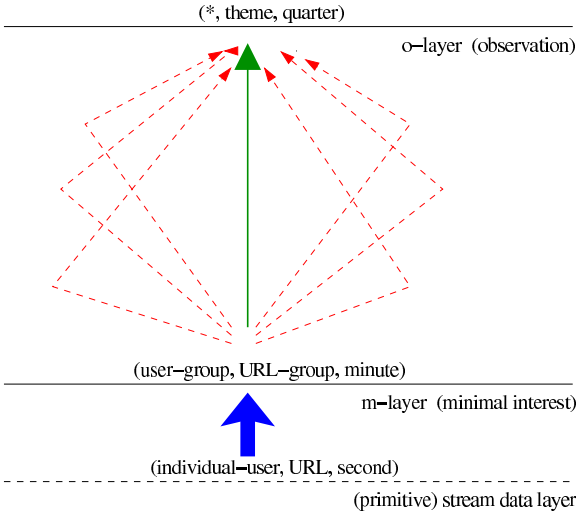


Figure 2: Two critical layers in the stream cube

Example 3. Assume that "(*individual_user*, *URL*, *second*)" forms the primitive layer of the input stream data in Ex. 1. With the *tilt time frame* as shown in Figure 1, the two critical layers for power supply analysis are: (1) the *m*-layer: (*user_group*, *URL_group*, *minute*), and (2) the *o*-layer: (*, *theme*, *quarter*), as shown in Figure 2.

Based on this design, the cuboids lower than the *m*-layer will not need to be computed since they are beyond the minimal interest of users. Thus the minimal regression cells that our base cuboid needs to be computed and stored will be the aggregate cells computed with grouping by *user_group*, *URL_group*, and *minute*. This can be done by aggregations (1) on two dimensions, *user* and *URL*, by rolling up from *individual_user* to *user_group* and from *URL* to *URL_group*, respectively, and (2) on time dimension by rolling up from *second* to *minute*.

Similarly, the cuboids at the *o*-layer should be computed dynamically according to the tilt time frame model as well. This is the layer that an analyst takes as an observation deck, watching the changes of the current stream data by examining the slope of changes at this layer to make decisions. The layer can be obtained by rolling up the cube (1) along two dimensions to $*$ (which means *all* user_category) and *theme*, respectively, and (2) along time dimension to *quarter*. If something unusual is observed, the analyst can drill down to examine the details and the exceptional cells at low levels. ■

2.2.3 *Partial materialization of stream cube*

Materializing a cube at only two critical layers leaves much room for how to compute the cuboids in between. These cuboids can be precomputed fully, partially, not at all (i.e., leave everything computed on-the-fly), or precomputing exception cells only. Let us first examine the feasibility of each possible choice in the environment of stream data. Since there may be a large number of cuboids between these two layers and each may contain many cells, it is often too costly in both space and time to fully materialize these cuboids, especially for stream data. Moreover, for the choice of computing *exception cells* only, the problem becomes how to set up an exception threshold. A too low threshold may lead to computing almost the whole cube, whereas a too high threshold may leave a lot of cells uncomputed and thus not being able to answer many interesting queries. On the other hand, materializing nothing forces all the aggregate cells to be computed on-the-fly, which may slow down the response time substantially. Thus, it seems that the only viable choice is to perform partial materialization of a stream cube.

Partial materialization of data cubes has been studied substantially in previous work [12, 5]. With the concern of both space and on-line computation time, we propose a "*popular path*" approach, which computes and maintains a single popular aggregation path from *m*-layer to *o*-layer so that queries directly on those (layers) along the popular path can be answered without further computation, whereas those deviating from the path can be answered with minimal computation from those reachable from the computed layers.

The details of this approach and its feasibility for stream data analysis will be examined in the next two sections.

3. STREAM CUBE COMPUTATION

Based on the above discussion, we examine the design and implementation of the popular-path approach for efficient computation of stream cubes.

First, based on the notion of *m*-layer, i.e., the minimal interesting layer, and the tilt time frame, stream data can be directly aggregated to this layer according to the tilt time scale. Then the data can be further aggregated following the popular path to reach the observation layer.

To facilitate efficient computation and storage of stream cube, a compact data structure needs to be designed so that the space taken in the computation of aggregations is minimized. A data structure, called H-tree, a hyper-linked tree structure introduced in [11], is revised and adopted here to ensure that a compact structure is maintained in memory for efficient computation of multi-dimensional and multi-level aggregations.

We present these ideas using an example.

Example 4. Suppose the stream data to be analyzed contains 3 dimensions, A, B and C, each with 3 levels of abstraction (excluding the highest level of abstraction "$*$"), as (A_1, A_2, A_3), (B_1, B_2, B_3), (C_1, C_2, C_3), where the ordering of "$* > A_1 > A_2 > A_3$" forms a high-to-low hierarchy, and so on. The minimal interesting layer (the *m*-layer) is (A_2, B_2, C_2), and the *o*-layer is $(A_1, *, C_1)$. From the *m*-layer (the bottom cuboid) to the *o*-layer (the top-cuboid to be computed), there are in total $2 \times 3 \times 2 = 12$ cuboids, as shown in Figure 3.

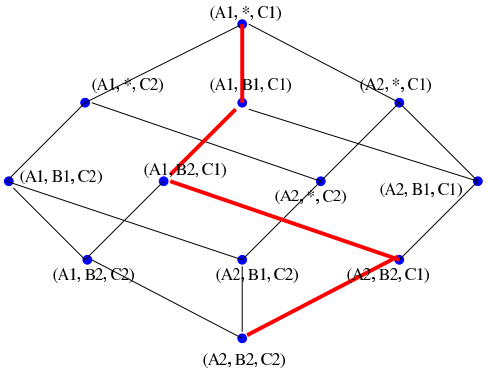


Figure 3: Cube structure from the m-layer to the o-layer

Suppose that the popular drilling path is given (which can usually be derived based on domain expert knowledge, statistical analysis or experiments). Assume that the given popular path is $\langle (A_1, C_1) \rightarrow B_1 \rightarrow B_2 \rightarrow A_2 \rightarrow C_2 \rangle$, shown as the dark-line path in Figure 3, where $\{A_1, C_1\}$ means that the ordering of A_1 or C_1 is unimportant. Then each path of an H-tree from root to leaf is ordered the same as the popular path.

This ordering generates a compact tree because the set of low level nodes that share the same set of high level ancestors will share the same prefix path using the tree structure. Each tuple, which represents the currently in-flow stream data, after being generalized to the *m*-layer, is inserted into the corresponding path of the H-tree. An example H-tree is shown in Fig 4. In the leaf node of each path, we store relevant measure information of the cells of the *m*-layer. The upper level measures are computed using the H-tree and its associated links.

An obvious advantage of the *popular path approach* is that the nonleaf nodes represents the cells of those layers (cuboids) along the popular path. Thus these nonleaf nodes naturally serves as the cells of the cuboids along the path. That is, it serves as a data structure for intermediate computation as well as the storage area for the computed measures of the layers (i.e., cuboids) along the path.

Furthermore, the H-tree structure facilitates the computation of other cuboids or cells in those cuboids. When a query or drill-down clicking requests to compute cells outside the popular path, one can find the closest lower level computed cells and use such intermediate computation results to compute the measures requested, because the corresponding cells can be found via a linked list of all the corresponding nodes contributing to the cells. ■

Based on this example, the popular path partial cube computation algorithm can be presented as follows.

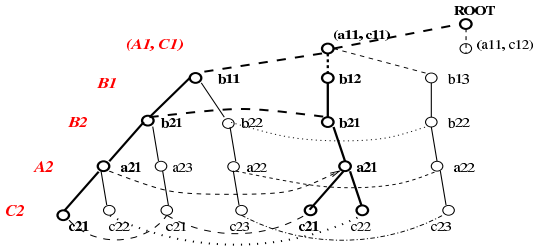


Figure 4: H-tree structure for cube computation

ALGORITHM 1 (**Popular-path**). *Computing cuboids along the* popular-path *between the m-layer and the o-layer.*

Input. *(1) multi-dimensional multi-level stream data, (2) the m and o-layer specifications, and (3) a given popular drilling path.*

Output. *All the aggregated cells of the cuboids along the popular path (and between the m- and o-layers).*

Method.

1. *Each tuple, which represents a minimal addressing unit of multi-dimensional multi-level stream data, is scanned once and generalized to the m-layer. The generalized tuple is then inserted into the corresponding path of the H-tree, increasing the count and aggregating the measure values of the corresponding leaf node.*
2. *Since each branch of the H-tree is organized in the same order as the specified popular path, aggregation is performed from the m-layer all the way up to the o-layer by aggregating along the popular path. The step-by-step aggregation is performed while inserting every new generalized tuple.*
3. *The aggregated cells are stored in the nonleaf nodes in the H-tree, forming the computed cuboids along the popular path.*

Analysis. *The H-tree ordering is based on the popular drilling path given by users or experts. This ordering facilitates the computation and storage of the cuboids along the path. The aggregations along the drilling path from the m-layer to the o-layer are performed during the generalizing of the stream data to the m-layer, which takes only one scan of stream data. Since all the cells to be computed are the cuboids along the popular path, and the cuboids to be computed are the nonleaf nodes associated with the H-tree, both space and computation overheads are minimized.* ■

Notice that the process discussed in the algorithm is essentially an incremental computation method, using the tilt time frame of Figure 1. Assuming that the memory contains the previously computed m and o-layers, plus the cuboids along the popular path, and stream data arrive every second. The new stream data are accumulated (by generalization) in the corresponding H-tree leaf nodes. Since the time granularity of the m-layer is minute, the aggregated data will trigger the cube computation once every minute, which rolls up from leaf to the higher level cuboids. When reaching a cuboid whose time granularity is quarter, the rolled measure information remains in the corresponding minute slot until it reaches the full quarter (i.e., 15 minutes) and then it rolls up to even higher levels, and so on.

Notice in this process, the measure in the time interval of each cuboid will be accumulated and promoted to the corresponding coarser time granularity, when the accumulated data reaches the corresponding time boundary. For example, the measure information of every four quarters will be aggregated to one hour and be promoted to the hour slot, and in the mean time, the quarter slots will still retain sufficient information for quarter-based analysis. This design ensures that although the stream data flows in-and-out, measure always keeps up to the most recent granularity time unit at each layer.

4. FEASIBILITY ANALYSIS

Here we present a brief feasibility analysis of our proposed approach, which shows that the total memory and computation time taken is small, in comparison with several other alternatives, and it is realistic to compute such a partially aggregated cube, and such precomputed cube will benefit OLAP analysis of stream data.

Limited by space, we report only two studies with synthetic data streams of various characteristics (see below); the records have a varied number of non-* dimension values and these values are at different levels.

The data stream is generated by a data generator similar in spirit to the IBM data generator [2] designed for testing data mining algorithms. The convention for the data sets is as follows: $D3L3C10T400K$ means there are 3 dimensions, each dimension contains 3 levels (from the m-layer to the o-layer, inclusive), the node fan-out factor (cardinality) is 10 (i.e., 10 children per node), and there are in total 400K merged m-layer tuples.

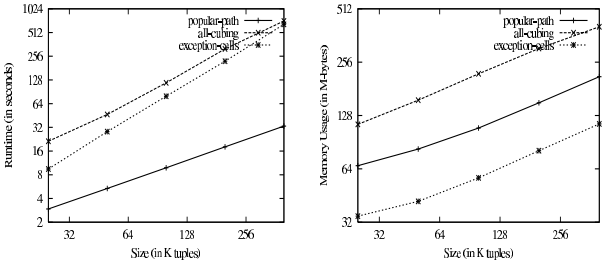
All experiments were conducted on a 750MHz AMD PC with 512 megabytes memory, running Microsoft/Windows-2000 Server. All the methods were implemented using Microsoft/Visual C++ 6.0.

Our design framework has some obvious performance advantages over some alternatives in a few aspects, including (1) *tilt time frame* vs. *full non-tilt time frame*, (2) *using minimal interesting layer* vs. *examining stream data at the raw data layer*, and (3) *computing the cube up to the apex layer* vs. *computing it up to the observation layer*. Consequently, our feasibility study will not compare the design that does not have such advantages since they will be obvious losers.

Since a data analyst needs fast on-line response, and both space and time are critical in processing, we examine both time and space consumption. In our study, besides presenting the total time and memory taken to compute and store such a stream cube, we compare the two measures (time and space) of the *popular path* approach against two alternatives: (1) the *all-cuboids* approach, i.e., materializing all the cuboids between the m- and o- layers, and (2) the *exception-cells* approach, i.e., materializing only the exception cells of the cuboids between the m- and o- layers, and we set exception threshold to be 1%, i.e., only top 1% cells will be considered as exceptions at each layer (cuboid).

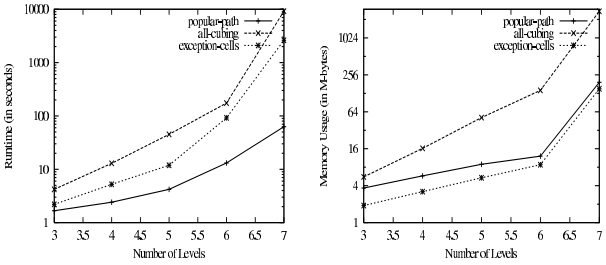
The performance results are reported in Figures 5–6.

Figure 5 shows processing time and memory usage used for the three approaches, with increasing size of the data set. Since *all-cuboids* computes all the cells from the m-layer all the way up to the o-layer, the total processing time is much higher than popular-path. However, since the exception rate is set to only 1% in our test, the *exception-cells* approach may take less memory than popular-path (but fails



a) Time vs. size b) Space vs. size

Figure 5: Feasibility study: Time and space vs. # tuples at the m-layer for the data set $D3L3C10T400K$



a) Time vs. # levels b) Space vs. # levels

Figure 6: Time and space vs. # of levels

to answer any queries which go beyond 1% exception rate). Nevertheless, it still takes much longer time to evaluate such cells since it has to compute almost the whole cube to derive such exceptions (although there are some optimization techniques which will not discussed here for lack of space).

Figure 6 shows the processing time and memory usage vs. the number of levels from m- to o- layers, with cube structure of $D2C10T10K$ and the exception rate at 1%. In all the approaches, with the growth of number of levels in the data cube, both processing time and space usage grow rapidly. However, popular-path grows much slower than *exception-cells* in both time and space and thus cost almost the same space as *exception-cells* at 1% exception rate when number of levels reaches 7.

From this study, one can see that popular-path is efficient and feasible method for computing multi-dimensional, multi-level stream cubes.

Finally, this performance study computes the cubes for the whole set of stream data. In practice, one just need to incrementally compute the newly generated stream data. Thus, the computation time should be substantially shorter than that shown here although the total memory usage may not reduce due to the need to store data in the layers along the popular path between two critical layers.

5. CONCLUSIONS

In this paper, we promote on-line analytical processing of stream data and proposed a feasible method for on-line computation of multi-dimensional, multi-level stream cube. The method uses a *tilt time frame*, explores *minimal interesting and observation layers*, and adopts an *popular path approach* for efficient computation and storage of stream cube to facilitate OLAP analysis of stream data. Our feasibility study shows that the method is cost-effective and is a realistic approach with current computer technology.

We believe this study is the first that proposes and explores on-line analytical processing of multi-dimensional, multi-level stream data. There are a lot of issues to be explored further. For example, we have implemented and studied an H-tree structure-based algorithm for computing popular-path stream cube. It is interesting to explore other cubing techniques, such as [14, 4], at popular-path cubing, as well as developing high performance algorithm for exploring the non-precomputed cube space. Moreover, we believe that a very important direction is to further develop data mining methods to take advantage of multi-dimensional, multi-level stream cubes for single-pass on-line mining for deep knowledge in stream data.

6. REFERENCES

[1] S. Agarwal, et al. On the computation of multidimensional aggregates. *VLDB'96*, pp. 506–521, Bombay, India, Sept. 1996.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. *ICDE'95*, pp. 3–14, Taipei, Taiwan, Mar. 1995.

[3] S. Babu and J. Widom. Continuous queries over data streams. *ACM SIGMOD Record*, 2001(3):109–120.

[4] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. *SIGMOD'99*, pp. 359–370, Philadelphia, PA, June 1999.

[5] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26:65–74, 1997.

[6] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continuous data streams. *SIGMOD'01*, pp. 13–24, Santa Barbara, CA, May 2001.

[7] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. *VLDB'01*, pp. 79–88, Rome, Italy, Sept. 2001.

[8] 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*, 1:29–54, 1997.

[9] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summeries. *SIGMOD'01*, pp. 58–66, Santa Barbara, CA, May 2001.

[10] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. *FOCS'00*, pp. 359–366, 2000.

[11] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg cubes with complex measures. *SIGMOD'01*, pp. 1–12, Santa Barbara, CA, May 2001.

[12] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. *SIGMOD'96*, pp. 205–216, Montreal, Canada, June 1996.

[13] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. *KDD'01*, pp. 71–80, San Fransisco, CA, Aug. 2001.

[14] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *SIGMOD'97*, pp. 159–170, Tucson, Arizona, May 1997.