Storage Estimation for Multidimensional Aggregates in OLAP

Kanda Runapongsa, Thomas P. Nadeau, and Toby J. Teorey
Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109, USA
E-mail: {krunapon, nadeau, teorey}@eecs.umich.edu

Abstract

On-line analytical processing (OLAP) is an important technique for analyzing data in decision support systems. Most analytical queries require aggregation of the interesting data. Pre-aggregation is one of the most important techniques used to speed up the query response time. However, precomputing every aggregate takes a large amount of time and space. The decision of which aggregates should be precomputed and how much space is required is thus important. By estimating the storage space required for each aggregate view, we can allocate the space for aggregates efficiently and decide which aggregates to precompute. We investigate four existing strategies for this problem: two based on mathematical approximations, one based on sampling, and one hybrid approach based on mathematical approximation and sampling. We propose a new hybrid strategy that is based on mathematical approximation and sampling and is easy to compute. We evaluate the accuracy of these algorithms in estimating the storage explosion due to aggregation for different data distributions and data densities. The results indicate that our proposed strategy approximates the explosion more accurately than other strategies.

1 Introduction

Most data analysis supported by On-Line Analytical Processing (OLAP) systems is multidimensional and hierarchical [19, 21]. The analytical queries used for making a decision usually require summarization and comparison. Therefore, these queries require the computation of several aggregate functions, such as *sum* and *count*, over large amount of multidimensional data. To meet the user's need for fast response time, all OLAP products precompute some aggregates. The more aggregates are precomputed, the faster queries can respond to the user. However, the pre-compute aggregation technique has a storage constraint. The database administrator needs to estimate how much storage a given set of pre-computed aggregates requires. The aggregation of the interesting data is usually kept in a *view*. A view is a *virtual relation* defined by a query expression [18]. A virtual relation is a relation that is not part of the conceptual model but is made visible to a user [18]. A view may be materialized-that is, stored physically [18]. The size of a given view is the number of distinct values of the attributes it groups by [11]. Many materialized view selection algorithms need to know the size of the candidate materialized view. In this paper we investigate some techniques that have been proposed for determining the space storage needed for a set of views. We compare these techniques with our new hybrid approach.

1.1 An Example

To clarify the problem we are considering, we begin with an example. Consider a table (a relation) of sales with the schema

```
Sale(ProductId, DayId, Quantity)
```

with the meaning that each tuple represents some quantity of some product sold on some

date. We assume that we have some information about products and times captured in the following tables.

`Product(ProductId, Category)` and
`Time(DayId, Week)`

From the above example, `Quantity` is referred to as a *measure* value because it is what we are measuring. `Product` table and `Time` table are referred to as *dimension* tables since each one of them represents the measure value along one perspective or one dimension. `Sale` table is referred to as a *fact* table. A fact table correlates all dimensions and contains information on the measure values. The attributes in dimension tables are usually hierarchical. For example, a particular category has a certain set of products.

The user of this data may want to know the total sales of each product that have been sold. We then need to aggregate the quantity of the sale of each product. The query can be written in the Structured Query Language (SQL) as follows:

```
select ProductId, SUM(Quantity)
from Sale
group by ProductId;
```

If the `Sale` table is large, the query response time will be long. However, if the aggregate of each product is precomputed, this query and queries derived from this aggregate can be answered very quickly. An example of other queries that can be derived from this aggregate is the query that asks for the total sales of all products. If we precompute every aggregate, the user query response time can be answered almost instantly. On the other side, storing all these precomputed aggregates may be infeasible. This is because we may not have enough space. More importantly, it may be very expensive to update all of these aggregates when the fact table is updated. Therefore, the task of the database administrator faces is to choose a set of aggregates to precompute and store. In this paper, we first consider the problem of estimating how much storage will be required if all possible combinations of dimensions and their hierarchies are precomputed. Once we have decided how to estimate this full precomputation, the extension to precomputation of a subset is straightforward.

The full precomputation of the data is usually done by using the *cube* operator which is proposed by [8]. The cube operator is the n-dimensional generalization of the SQL group by operator. The cube on n attributes computes the group by aggregates for each possible subset of these dimensions. In our example, this is: (), (ProductId), (DayId), and (ProductId, DayId). When we consider the possibility of aggregating over hierarchies, we get a generalization of the cube which we will refer to as the cube from here on. The cube as defined by [8] is referred to as a cube without hierarchies. In our example, the cube with hierarchies computes additional aggregates for: (Category), (ProductId, Week), (Category, DayId), (Category, Week), and (Week).

Estimation of the size of the view, consisting of the aggregation of data, without actually materializing the view is the problem that we are solving. Computing the cube usually takes a long time. For example, the *Overlap* method which was proposed by [1] requires multiple scans and sorts of the input table. Moreover, computing the cube for a schema with a large number of dimensions or a large number of hierarchical levels requires computing many aggregates. For example, a schema with 5 dimensions where each dimension has 4 levels requires us to compute $(4+1)^5 = 3125$ distinct SQL group by queries.

1.2 Paper Organization

The remainder of this paper begins by the related work in Section 2. Then, we present factors that affect the cube size in Section 3. Section 4 presents various algorithms for approximating the cube size. Section 5 illustrates the experimental results and comparisons between these different techniques. Section 6 summarizes what we have learned about these algorithms. Finally, Section 7 discusses future work.

2 Related Work

The problem addressed in this paper is closest to the work in [17], which surveyed the available

approaches and proposed an algorithm that applied the probabilistic counting algorithm [6] to estimate the size of the cube. The solution of the problem in this paper is used in the materialized view selection algorithms. [11], [2], [10], [9], [20], and [16] addressed the materialized view selection problem where the goals are to answer all the interesting queries while minimizing the total query evaluation and view maintenance cost. This problem is also known as the Data Warehousing (DW) configuration problem.

A related estimation problem is to estimate the number of distinct values of an attribute in a relation. [12] is the first paper that presented the sampling-based estimator which explicitly takes into account the degree of skew in the data. Although the probabilistic counting algorithm proposed by [17] gives us an accurate approximation of the size of the cube, it requires that each tuple in the relation is scanned and processed. Moreover, if we have the addition of new data, this may change the size of the cube. According to the probabilistic counting algorithm, we need to load bitmaps that store the statistics of the previous data and rescan the whole table. If the table is very large and there are many updates, this yields a long execution time. [12] also argued that as databases grew in size, this exhaustive processing became increasingly undesirable. The sampling and analytical techniques that were proposed by [12] is quite complicated. It requires us to compute about 6 complex equations besides performing an approximate χ^2 test for uniformity.

Most recently, [13] proposed a novel approach for the multidimensional selectivity estimation. This method maintains compressed information from a large number of small-sized histogram buckets by using the discrete cosine transform. The experimental results of this algorithm show that this algorithm has low error rates and low storage overheads even in high dimensions. We are currently investigating this algorithm in detail.

We are interested in developing a simple algorithm that gives us reasonable accuracy. We propose an algorithm that is based on mathematical approximation and random sampling.

3 Cube Size Parameters

The cube size here is the sum of the number of tuples in the views consisting of possible grouping attributes. To estimate the cube size accurately, we develop a set of parameters that affect the cube size. Along with these parameters, we also give the examples of how these parameters affect the cube size. The parameters are: the number of dimensions, the number of hierarchical levels, the attribute size, the degree of data density, and the degree of data skew. A description of each parameter follows:

3.1 Number of Dimensions and Number of Hierarchical Levels

The number of dimensions is the number of tables that have primary keys as foreign keys in the fact table. The number of hierarchical levels is the number of attributes that are functionally dependent on primary attributes in each dimension. As the number of hierarchical levels or the number of dimensions increases, the number of views consisting of possible group bys also increases. This is shown in the following equation.

Let l_i be the number of hierarchical levels in dimension i, nv be the number of aggregate views, and N be the number of dimensions. Then,

$$nv = \prod_{i=1}^{N} (l_i + 1) \qquad (1)$$

In our sales example, the sale data can also be viewed conceptually as a two-dimensional array with hierarchies on the dimensions. Tables 1, 2 and 3 are three samples of the sales data sets. The **x**'s are sales quantity; entries that are blank correspond to (ProductId,DayId) combinations for which there are no sales. Table 4 shows the variation in size of the cube of the three different data sets that are shown in Tables 1, 2, and 3.

Table 1: Database DB1

		Book			Coat		
		P1	P2	P3	P4	P5	P6
Week1	D1	x				x	
	D2			x			
	D3		x				
	D4				x		
	D5			x			
	D6						
	D7				x		
Week2	D8						x
	D9	x					
	D10		x				
	D11					x	x
	D12	x					
	D13		x				
	D14			x			

Table 2: Database DB2

		Book			Coat		
		P1	P2	P3	P4	P5	P6
Week1	D1	x	x	x			
	D2	x	x	x			
	D3	x	x	x			
	D4						
	D5						
	D6						
	D7						
Week2	D8				x	x	x
	D9				x	x	x
	D10						
	D11						
	D12						
	D13						
	D14						

Table 3: Database DB3

		Book			Coat		
		P1	P2	P3	P4	P5	P6
Week1	D1	x				x	
	D2			x			x
	D3	x	x				
	D4				x		
	D5			x		x	
	D6						
	D7		x		x		
Week2	D8						x
	D9	x					
	D10		x				
	D11			x		x	x
	D12	x					
	D13		x		x		
	D14			x		x	

Table 4: The variation in size of the cube of three different databases

Group By	DB1	DB2	DB3
()	1	1	1
(ProductId)	6	6	6
(Category)	2	2	2
(ProductId,DayId)	15	15	22
(ProductId,Week)	10	6	12
(Category,DayId)	14	5	20
(Category,Week)	4	2	4
(DayId)	13	5	13
(Week)	2	2	2
Size of Cube	67	44	82

The size of the cube without hierarchies on DB1 would be the sum of the sizes of the following group bys: (), (ProductId), (DayId), and (ProductId, DayId). This sum is 35 tuples. On the other hand, the cube with hierarchies as shown in Table 4 has 67 tuples. Although DB1 and DB2 has the same number of the tuples in a base table (ProductId,DayId), the sizes of the cubes in these two databases are significantly different.

3.2 Attribute Size

Each aggregate view is defined by a query that is grouped by a set of attributes.

Let V be a view that consists of attributes $a_1, a_2, ..., a_k$ where k be the number of attributes in the view V, $n(a)$ be the cardinality of the attribute a, and $ms(V)$ be the maximum size of view V. Then,

$$ms(V) = \prod_{i=1}^{k} n(a_i) \qquad (2)$$

Returning to our previous database example, we may have the following query that responds to aggregate G that groups by `Product.Type` and `Time.Week`.

```
select SUM(Quantity), Product.Type, Time.Week
from Sale, Product, Time
where Sale.ProductId = Product.ProductId
and Sale.TimeId = Time.TimeId
group by Product.Type, Time.Week;
```

The maximum size of view V that consists of the aggregate G in DB1 is

$$\begin{aligned} ms(V) &= n(Product.Type) \times n(Time.Week) \\ &= 2 * 2 = 4 \end{aligned}$$

From Eq. (2), as the dimension size increases, the maximum size of the view also increases.

3.3 Degree of Data Density

According to [3], "a data set is said to be *sparse* if most points in the attribute space defined have no data points corresponding. Conversely, a data set is *dense* if most coordinate points in the attribute space have at least one data point defined." As shown in Table 4, the dense data (DB3) has a larger cube size than the sparse data (DB1). The denser data is, the greater size of the cube is. To formally find the degree of data density, we let F be a fact table, k_i be a primary key of dimension i, $n(k_i)$ be the cardinality of the attribute(s) that form the primary key k_i, $ms(F)$ be the maximum size of the fact table F, and N be the number of dimension tables. Then,

$$ms(F) = \prod_{i=1}^{N} n(k_i) \tag{3}$$

Formally, data density can be defined as follows: let $|F|$ be the number of tuples in the fact table and dd be data density. Then,

$$dd = \frac{|F|}{ms(F)} \tag{4}$$

$$dd = \frac{|F|}{\prod_{i=1}^{N} n(k_i)}$$

3.4 Degree of Data Skew

According to [3], a data set is said to be *skewed in frequency* if the number of data points per attribute point has a high variance across the entire data space, but has a substantially lower variance in appropriately defined "local" regions. On the other hand, a data set is *skewed in value* if the attribute value for a small number of data points differs substantially from the attribute value for the bulk of the population. Note that skew in value implies skew in frequency over the attribute value range, however, the converse is not true. In this paper, "skew" refers to "skew in frequency". As shown in Table 4, the skewed data (DB2) has a smaller cube size than the sparse data (DB1) although both DB1 and DB2 have the same base data size.

Data in real life is often non-uniformly distributed as shown in [23]. For example, James Joyce's novel *Ulysses*, with its 260,430 running words, has only 29,399 different words. The 10th most frequent word (r = 10) occurs 2,653 times (f = 2,653); and the 100th most frequent word (r = 100) occurs 265 times (f = 265). According to [23] model, the frequency of some event (P), as a function of rank (r) is a power-law function

$$P(r) \approx \frac{C}{r^{\theta}} \tag{5}$$

with the exponent θ close to unity.

We define the degree of skew as the value of θ. When $\theta = 1$, degree of skew is 1. This means that data is non-uniformly distributed. When $\theta = 0$, we have uniformly distributed data. That is, all cells with different ranks have the same probability of receiving a valid data value.

Let $|F|$ be size of the fact table F, $ms(F)$ be the maximum size of the fact table F, v be the value of the combination of dimension attributes in the fact table, and $P(v)$ be the probability that the value v will appear in the fact table. Let all the values be ranked in the order of their popularity where value v is the v'th most popular value. Then,

$$\begin{aligned} \sum_{v=1}^{ms(F)} P(v) &= |F| \\ \sum_{v=1}^{ms(F)} \frac{C}{v^{\theta}} &\approx |F| \end{aligned}$$

$$C \approx \frac{|F|}{\sum_{v=1}^{ms(F)} \frac{1}{v^{\theta}}} \tag{6}$$

where θ is in the range of 0 and 1. Note that the Zipf distribution is one of many different proposed models for data skew. For example, [5] proposed modeling skewed distribution using multifractrals and the '80-20' law.

4 Approximating the Cube Size

First we describe the mathematical approximations. Their advantages are that they are simple and fast. However, they do not work for skewed data. The next algorithm that we look at is the linear sampling algorithm. Unlike the mathematical approximation which considers only the database schema, the linear sampling method also looks at data distribution. However, the linear sampling method assumes that the more tuples we add to the fact table, the more tuples the cube has. This is not true especially when the data in the fact table is already dense. In the next algorithms, the assumption is changed to that the larger the estimated size of the view is, the larger the actual size of the view is. The estimated view size is obtained from using the mathematical approximation. The combination of using the sampling and the mathematical approximation is a hybrid approach. A description of each approach follows:

4.1 Mathematical Approximation Algorithms

Let $ms(F)$ be the maximum size of a fact table F, $|F|$ be the size of the fact table F, $ms(V)$ be the maximum size of a view V, and $es(V)$ be the estimated size of the view V.

[7] proposed to estimate the size of the view V by using Yao's formula [22] as follows:

$$es(V) = ms(V) \times \left[1 - \prod_{i=1}^{|F|} \frac{ms(F) \times d - i + 1}{ms(F) - i + 1}\right] \quad (7)$$

where $d = 1 - \frac{1}{ms(V)}$.

When $\frac{ms(F)}{ms(V)}$ is sufficiently large, this formula is well approximated by Cardenas' formula [4] as follows:

$$es(V) = ms(V) \times \left(1 - \left(1 - \frac{1}{ms(V)}\right)^{|F|}\right) \quad (8)$$

Note that

$$\text{if} \quad |F| > ms(F) - \frac{ms(F)}{ms(V)}$$
$$\text{then} \quad es(V) = ms(V)$$

The size of the cube on the data in the fact table F, $CUBE(F)$, is defined as follows:

$$CUBE(F) = \sum es(V_i)$$

where $V_i = V_1, V_2, ..., V_n$ and n is the number of possible aggregate views that are defined by possible grouping attributes in the fact table F.

Cardenas' formula and Yao's formula are based on the assumption that data is uniformly distributed. To apply these two methods, we need to know the number of distinct values for each attribute in the relation. The system catalog typically maintains these statistics. Using Eq. (2) and Eq. (3), we can obtain $ms(V)$ and $ms(F)$ respectively. $|F|$ can be obtained by counting the number of tuples in the fact table.

Any skew in the data tends to reduce the size of the aggregate view. Hence the uniform assumption tends to overestimate the size of the views and thereby also overestimates the size of the cube. The mathematical approximation methods do not take the degree of skew into account, thus they give the crude estimation. However, these methods have the advantage that they are simple and fast.

4.2 A Linear Sampling Algorithm

Throughout this paper, we use sampling without replacement to avoid having duplicate tuples in the table. Selection without replacement also minimizes estimation error [12]. Specifically, in generating the sample data, for each tuple in the fact table, we want to accept it to a sample of the fact table with the probability equal to the sample fraction. Starting with the first tuple, we called `rand` function in C to generate a random integer between 0 and `RAND_MAX`. We then multiply the generated random integer with 100 and divide by `RAND_MAX`. If the result of this computation is less than or equal to the sample fraction, we then check

whether this tuple is already included in the sample of the fact table. If it is not, we then include this tuple in the sample and mark this tuple as one already included in the sample. Otherwise, we go to the next tuple and generate a new random integer. We repeatedly follow this procedure until we get the number of tuples in the sample as the number of tuples in the original fact table multiplied with the sample fraction.

[17] proposed to use this approach to estimate the size of the cube. This algorithm takes a random subset of the database, and computes the cube on that subset. Then it scales up this estimate by the ratio of the data size to the sample size.

Let $|F|$ be the size of the fact table F, $CUBE(F)$ be the size of the cube on the entire fact table F, $|s|$ be the size of the sample s, and $CUBE(s)$ be the size of the cube computed on the sample s. Then, we can approximate $CUBE(F)$ by

$$CUBE(F) = CUBE(s) \times \frac{|F|}{|s|}$$

This approximation is also very crude. The advantage of this computation is simple. The advantage over the uniform assumption is gained by examining a statistical subset of the database. However, this also comes with the disadvantage over the uniform assumption estimates because it takes longer to compute.

4.3 Mathematical Approximation with Sampling Algorithms

The key idea of the solutions in this section is based on a motivation to perceive the skewness of the data while keeping the algorithm simple. Sampling gives us some information about the data distribution. The problem with linear scaling is that it assumes that the number of tuples in the cube always increases as the number of tuples in the fact table increases. This is not true especially when data in the fact table is dense. A better assumption is that the actual size of the view is proportional to the estimated size of the view. [15] proposed the proportional skew effect approach. This approach first obtains an initial estimate of the view size using Cardenas' formula. To refine this estimate, we get the actual view size of a sample of the fact table. We then compare the actual size and the estimated size of the view based on the sample data to get a correction factor. We then take this factor to refine the estimated size of the view based on the entire fact table. In this paper, we propose to bound the estimated size of each view to either the maximum size of that view or to the size of the fact table.

4.3.1 Proportional Skew Effect Algorithm (PSE)

Steps 1 and 2(a)-2(d) of this approach were proposed in [15].

1. Select a random sampling s from a fact table F.
2. For each view V
   (a) Use Cardenas' formula to estimate the size of the view V, $es_F(V)$, based on data in the fact table F. That is,

$$es_F(V) = ms(V) \times \left(1 - \left(1 - \frac{1}{ms(V)}\right)^{|F|}\right)$$

   where $ms(V)$ can be computed from using Eq.(2).

   (b) Use Cardenas' formula to estimate the size of the view V, $es_s(V)$, based on the sample s. That is,

$$es_s(V) = ms(V) \times \left(1 - \left(1 - \frac{1}{ms(V)}\right)^{|s|}\right)$$

   (c) Use SQL group by operator to obtain the actual size of the view V based on the sample s, $s_s(V)$.

   (d) Adjust the estimated size of the view V, $es'_F(V)$.

$$es'_F(V) = s_s(V) \times \frac{es_F(V)}{es_s(V)}$$

   (e) Bound the estimated size of the view V to either the actual size of the

fact table, $|F|$, or the maximum size of the view V, $ms(V)$. The final estimated size of the view V based on the fact table F, $es''_F(V)$, is:

$$es''_F(V) = min(es'_F(V), |F|, ms(V))$$

3. Total up the estimated size of each view to find the estimated size of the cube on data in the fact table F, $CUBE(F)$, as follows:

$$CUBE(F) = \sum es''_F(V_i)$$

where $V_i = V_1, V_2, ..., V_n$ and n is the number of possible grouping attributes in the fact table F.

4.3.2 Sample Frequency Algorithm (SF)

We notice that different combinations of attribute values have different frequencies appearing in the fact table. The appearance frequencies of different values become larger when the data is skewed. We use the appearance frequency in scaling up the distinct values in the sample to the distinct values in the entire fact table. We also use the error in estimating the size of the view based on the sample data to adjust the estimated size of the view based on the entire fact table.

The description of the algorithm is as follows:

Steps 1, and 2(a)-(c) are the same as ones in the proportional skew effect algorithm. Steps 2(d)-(e) of this algorithm are developed as follows:

Step 2(d): Adjust the Estimated Size of the View

First, we find the relative error in estimating the size of view V based on the sample s, $error_s(V)$.

$$error_s(V) = \frac{es_s(V) - s_s(V)}{s_s(V)}$$

where $es_s(V)$ is the estimated size of the view V based on the sample s and $s_s(V)$ is the actual size of the view V based on the sample s.

Then, we find the relative error, $error_s(V)$, to adjust the new estimated size of the view V based on the fact table F. We assume that the relative error is proportional to the data size. If we overestimate (underestimate) the size of the view based on the sample, we subtract (add) the estimated size of the view based on the fact table with the adjusted error to get the new estimated size of the view based on the fact table. We set the maximum estimated size of the view V based on the fact table F to the minimum between the size of the fact table F, $|F|$, and the maximum size of the view V, $ms(V)$. The minimum size of the view V based on the fact table F is the actual size of the view V based on the sample s, $s_s(V)$. The following describes our adjustment to the new estimated view size:

$es_F(V) = es_F(V) - error_s(V) \times es_F(V)$
if $error_s(V) > 0$
$es'_F(V) = max(es_F(V), s_s(V))$
else
$es'_F(V) = min(es_F(V), ms(V), |F|)$

Step 2(e): Estimate the Number of Distinct Values Uncovered in the Sample

The sample does not cover all distinct grouping attribute values in the entire fact table. The estimated number of distinct values that do not appear in the sample s is the difference between the new estimated size of the view V based the fact table F and the actual size of the view V based on the sample. That is, the estimated number of distinct values of attributes in view V that are uncovered by the sample s but are in fact table F is $es'_F(V) - s_s(V)$.

For $1 \leq i \leq |s|$, let f_i be the number of grouping attribute values that appear exactly i times in the sample s. Thus, $\sum_{i=1}^{|s|} f_i = s_s(V)$ and $\sum_{i=1}^{|s|} i f_i = |s|$. The number of values that appear i times in the total $s_s(V)$ distinct values is f_i. The estimated number of values that are excluded from the sample s but appear i times in the fact table F is $\frac{f_i}{s_s(V)} \times (es'_F(V) - s_s(V))$. We assume that large values appear with low frequency and small values appear with high frequency. We thus adjust the estimated number of values that appear i times in the fact table F, but not in sample s, as $f'_i = \frac{f_i}{s_s(v)} \times (es'_F(V) - s_s(V)) \times \frac{1}{i}$.

To find the estimated number of the distinct values in the view V based on the fact table F, we need to add the number of the distinct values in the sample with the estimated number of the distinct values that we do not see in the sample. Thus, the final estimated size of the view V, $es''_F(V)$, is defined as follows:

$$es''_F(V) = s_s(V) + \sum_{i=1}^{|s|} \frac{f_i}{s_s(V)} \times (es'_F(V) - s_s(V)) \times \frac{1}{i}$$

5 Experimental Results

In this section, we compare the performance of different estimation algorithms described in Section 3. First, we present the test database schema. Then, we describe our data set generation methods. Finally, we compare the estimated view sizes and cube sizes by various estimators. We also compare the relative errors in different estimation algorithms.

5.1 Test Database Schema

The relational OLAP (ROLAP) schema for the test database is below:

- **fact**(*d0* int, *d1* int, volume float);
- **dim0**(*d0* int, h01 string, h02 string);
- **dim1**(*d1* int, h11 string, h12 string);

The h**X**1 and h**X**2 attributes of all the dimension tables are uniformly distributed. The h01 values are drawn from 200 distinct values and the h02 values are drawn from 50 distinct values. The h11 values are drawn from 500 distinct values and the h12 values are drawn from 100 distinct values. Table 5 summarizes the database configuration.

Table 5: Small Database Configuration

Dimension Number	Dimension	Hierarchy	
		1	2
0	1000	200	50
1	1000	500	100

The sample fractions in the experiments are 10% and 15% of the size of the fact table. The degrees of skew are 0 (uniformly distributed), 0.2, 0.4, 0.6, 0.8, and 1 (highly skewed). Table 6 illustrates the relationship between data density and the number of tuples in the fact table.

Table 6: Data Density vs. Fact Table Size

Data Density (%)	Number of Tuples in the Fact Table
0.1	1000
0.5	5000
1	10000
5	50000

5.2 Experimental Setup

To understand how we generated the data it is easiest to think in terms of the array representation. To generate a uniform data set x% dense, we iterated through all cells of the array, as each cell was visited, we generated a random number between 0 and 1; if the random number r was in the range $0 \leq r \leq x/100$, we added the tuple that corresponds to this cell to a fact table, otherwise we did not add this tuple to the fact table.

For the Zipfian data we again iterated through all the cells of the array. We assumed that different cells have different ranks. The first cell that we visited had the first rank. The next right cell had the second rank, and so on. This time, each cell has a different probability of receiving a valid data value. The probabilities varied following a Zipfian distribution. Specifically, given that we know the degree of skew, θ, and the number of tuples in the fact table, $|F|$, we then use Eq. (6) to get C and plug C in Eq. (5) to get the probability of receiving a valid data value for each cell. The mapping from the distinct values in a dimension to its hierarchies uses a uniform distribution.

We get the actual number of tuples in each view by writing a Pro*C program using Oracle Precompiler on Sun Solaris 5.6. In this program, we do not actually create view. We use SELECT operator and GROUP BY operator to select all attributes in the view, and apply `sum` function to a measure value. We then look at the number of tuples selected in

the `sqlca.sqlerrd[2]`.

5.3 Performance Results

Table 7 depicts the relationship between the codes used in the graphs and the approaches that we carried out in the experiments. We do not show the experiments on Yao's formula because the estimates based on Yao's formula are very close to the estimates based on Cardenas' formula.

Table 7: Experimental Approaches

Approach	Code
1. **Mathematical approximation**	
1.1 Cardenas' formula [4]	cardenas
2. **Sampling-based**	
2.1 Linear Sampling [17]	sample
3. **Hybrid**	
3.1 Proportional Skew Effect [15]	PSE
3.2 Sample Frequency	SF
4 **Actual computing the cube**	actual

5.3.1 Estimated View Size Comparison

Figures 1 and 2 show for the actual sizes of the views and the estimated sizes of the views when data density = 1%, sample fraction = 10%, and degrees of skew = 0 and 1 respectively.

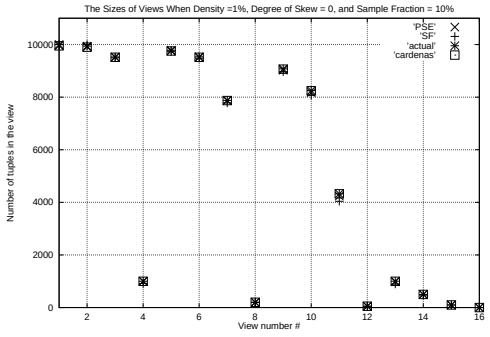

Figure 1: **Estimates vs. the actual sizes of the views when density = 1%, degree of skew = 0, and sample fraction = 10%**

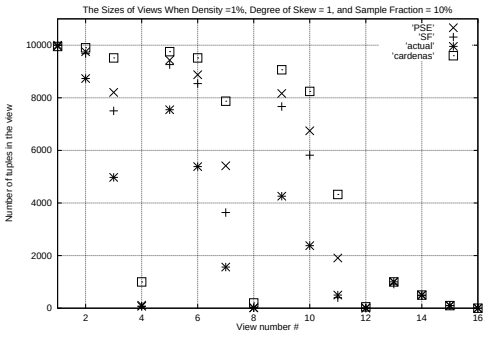

Figure 2: **Estimates vs. the actual sizes of the views when density = 1%, degree of skew = 1, and sample fraction = 10%**

Table 8: The Relation between the View Number and its Definition

View	Group by	View	Group by
1	d0, d1	9	h02, d1
2	d0, h11	10	h02, h11
3	d0, h12	11	h02, h12
4	d0	12	h02
5	h01, d1	13	d1
6	h01, h11	14	h11
7	h01, h12	15	h12
8	h01	16	-

As shown in Figure 1 where data is uniformly distributed, all algorithms estimate the view sizes accurately. This is because the mathematical approximation (cardenas) is based on the assumption that data is uniformly distributed. The actual view size and the view size estimated using the mathematical approximation are thus very close. The proportional skew effect algorithm (PSE) and the sample frequency algorithm (SF) use the difference between the view size estimated using the mathematical approximation based on the sample, and the actual view size based on the sample, to adjust the estimated size of view based on the entire fact table. Therefore, the estimates obtained from these algorithms are also accurate.

In Figure 2 where data is very skewed, the mathematical approximation overestimates the sizes of most views. This is because data is not uniformly distributed. There are many dupli-

cate values of the grouping attributes that the mathematical approximation does not see since it only looks at the database schema. When data is very skewed, the amount of the error in other algorithms also increases. Since the proportional skew effect approach and the sample frequency approach use the mathematical approximation to estimate the view size, these two algorithms are affected by the error in the mathematical approximation.

5.3.2 Estimated Cube Size Comparison

We also make the comparisons between the estimated cube size and the actual cube size. In addition to the algorithms that estimate the view size, we include the linear sampling (sample) as one of the algorithms that estimate the cube size.

Figure 3 shows for varying degrees of skew, the estimates and the actual sizes of the cubes when sample fraction = 10% and data density = 1%.

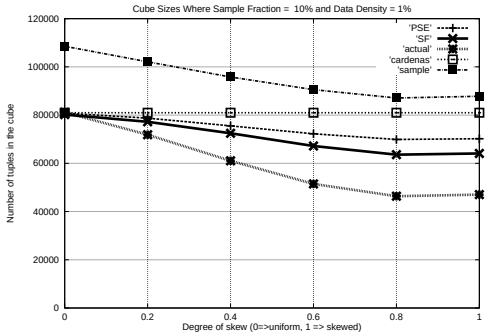

Figure 3: **Estimates vs. the actual sizes of the cube where sample fraction = 10% and data density = 1%**

As shown in Figure 3, the mathematical approximation closely estimates the cube size when data is uniformly distributed. However, as data becomes skewed, it overestimates the cube size. The mathematical approximation is independent of the underlying data distribution. Since the mathematical approximation is based on the assumption that data is uniformly distributed, its estimate is close to the actual size when the data is indeed uniformly distributed. Since the proportional skew effect approach and the sample frequency approach use the mathematical approximation in estimating the view size and the cube size is the sum of the view sizes, these two approaches also perform well as the mathematical approximation perform well. The linear sampling approach does not estimate the cube size accurately when data is uniformly distributed because it assumes that the cube size grows proportionally to the fact table size.

5.3.3 Analysis Performance

While the mathematical approximation always gives the same estimate no matter how data distribution is, the linear sampling approach, the proportional skew effect approach, and the sample frequency approach adjust their estimates as data distribution changes. Of these three approaches, the sample frequency approach has its estimate closest to the actual cube size. The linear sampling approach overestimates more than the mathematical approximation because it assumes that the cube size grows proportionally to the fact table size. Moreover, it does not see enough duplicates in skewed data. The proportional skew effect approach estimates more accurately than the linear sampling approach because it scales on each view instead of on the whole cube and it also puts the limit of the estimated size of each view to be either the maximum size of each view or the size of the fact table. The sample frequency approach estimates more accurately than the proportional skew effect approach, particularly when data is highly skewed. This is because the sample frequency approach exploits the information about the number of duplicates of different values in the sample data. On the other hand, the proportional skew effect approach does not exploit this information. Figures 4 and 5 show the actual cube sizes and their estimates for varying data densities where degrees of skew are fixed to 0 and 1 respectively.

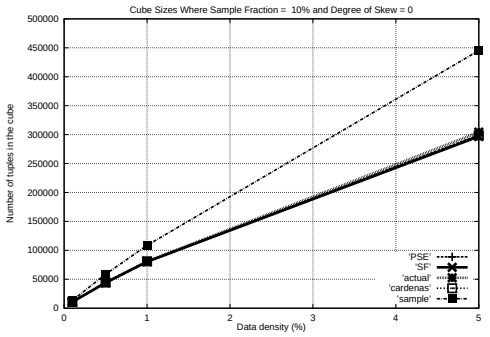

Figure 4: **Estimates vs. the actual sizes of the cube where degree of skew = 0**

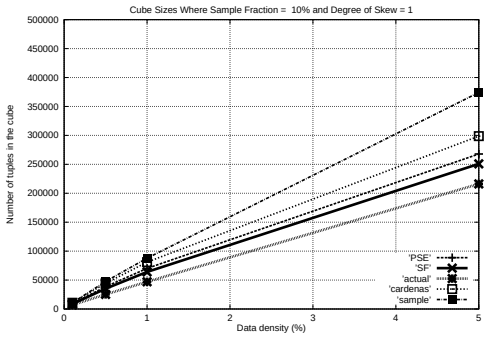

Figure 5: **Estimates vs. the actual sizes of the cube where degree of skew = 1**

From Figures 4 and 5, we observe that the actual cube size is almost proportional to data density. Its estimate is also almost proportional to data density. As shown in Figure 4, all algorithms except the linear sampling approach estimate the cube size accurately when data is uniformly distributed over all densities. On the other hand, as shown in Figure 5 where data is skewed, the gap between the actual cube size and its estimate becomes wider as data becomes denser. This is because the ratio of the duplicate values to all values increases in skewed and sparse data as data density increases. The sample fraction is not large enough to cover these duplicate values. Thus, all algorithms that use sampling tend to overestimate the number of distinct values in skewed data.

5.3.4 Relative Error Measurement

We measure the relative error in approximation made by different algorithms as follows: let S be the actual size of the cube, $S(A)$ be the estimated size of the cube by an algorithm A, and e_A^{rel} be the relative error in the algorithm A. Then,

$$e_A^{rel} = \frac{|S - S(A)|}{S} \text{ for } S(A) > 0$$

Figures 6 show the relative errors in different algorithms when the sample fractions are 10%.

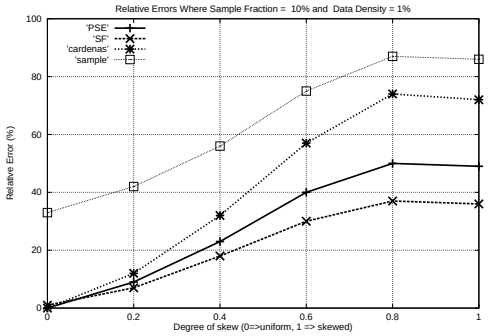

Figure 6: **The relative errors where sample fraction = 10%**

As shown in Figures 6, all algorithms except the linear sampling approach perform well for uniformly distributed data. However, as data becomes skewed, the error increases in all algorithms. This is because the mathematical approximation assumes that data is uniformly distributed and the sample fraction is not large enough to cover many duplicate values.

Figure 7 shows the average relative errors in different algorithms over different degrees of skew when sample fractions are 10% and 15%.

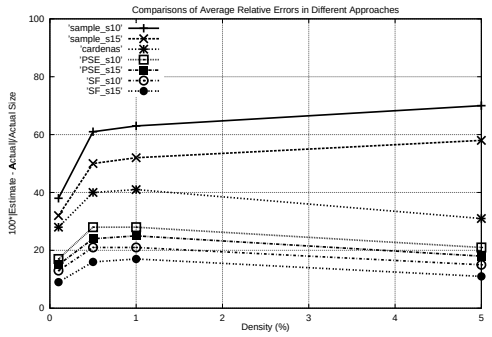

Figure 7: **The errors in different techniques using different sample fractions**

As shown in Figure 7, the sample frequency approach has the smallest error compared to other strategies. We observe that the error in the linear sampling method decreases more than the error in the hybrid approaches as the sample fraction increases. This is because the hybrid approaches use the mathematical approximation to estimate the view size. Therefore, increasing only the sample fraction does not improve the accuracy in the hybrid approaches as much as in the linear sampling approach. The accuracy of the hybrid approaches and the linear sampling approach depend on not only the quantity of the sampling fraction, but also the quality of the sampling fraction. We also observe that when data density increases up to 5%, the relative errors in the algorithms that use the mathematical approximation slightly decrease. This is because the error in estimating the distinct values in skewed and dense data becomes less significant than in skewed and sparse data. Dense data is likely to include many distinct values in skewed data. Therefore, when data is dense, the number of distinct values in skewed data is not much different from one in uniformly distributed data. The average error in estimating the number of distinct values in dense data for different data distributions thus decreases.

6 Conclusion

Precomputing aggregates on some subsets of dimensions and their corresponding hierarchies is the dominant technique to reduce the response time to queries. However, to decide which aggregates should be precomputed requires the estimated aggregate view size. In the experiments we carried out, our hybrid strategy estimates the view size and the cube size more accurately than other strategies, especially when data is skewed. Compared among the hybrid approaches, the sample frequency approach yields more accurate estimates than the proportional skew effect approach.

Comparing the algorithms based on their accuracy, we find that the linear sampling approach greatly overestimates the size of the cube. This estimate is strongly dependent on the number of duplicates present in the database, and the quantity and the quality of the sample. The mathematical approximation based on uniform distribution of the data, works well if the data is uniformly distributed. However, as sparse data becomes skewed, the algorithm is considerably inaccurate. On the other hand, the hybrid methods approximate skewed data more accurately than the mathematical approximation. The hybrid methods combine the simplicity and quickness of the mathematical approximation with the advantage of the sampling method which gives us some sense of data distribution.

However, all of hybrid approaches need to spend more time in aggregating the sample data. Also, the accuracy of the hybrid approaches depend on the accuracy of the estimated view sizes that are based on the mathematical approximation. Therefore, if these estimates have high error, the estimated values by the hybrid approaches also have high error.

7 Future Work

We would like to explore other estimation algorithms and compare the efficiency and accuracy of different algorithms. The algorithms that we are particularly interested in is the technique that uses a multi-dimensional histogram to estimate the size of a query that involves multiple attributes [14], the technique that uses wavelet, and the technique that uses compressed histogram information [13]. We also would like to the estimate approaches on real-world data

sets as well as TPC-D benchmarks.

Acknowledgements

The authors would like to thank Hidetoshi Uchiyama, Tracy Mullen, and Seksan Kiatsupaibul for earlier stimulating discussions and questions. The authors also would like to thank Nandit Soparkar, Atul Prakash, and Brian Noble for their valuable questions and comments. The authors thank as well to anonymous CASCON referees for pointing out some related work and for many suggestions that improved the paper.

About the Authors

Kanda Runapongsa is a Ph.D. student in the Department of Electrical Engineering and Computer Science at the University of Michigan at Ann Arbor. She received her B.S. degree in Electrical and Computer Engineering (1997) from Carnegie Mellon University and her M.S. degree in Computer Science and Engineering (1999) from the University of Michigan. Her research interests include data warehousing, OLAP, data mining, and query optimization.

Thomas P. Nadeau is a Ph.D. student in the Department of Electrical Engineering and Computer Science at the University of Michigan at Ann Arbor. He received his B.S. degree in Computer Science (1981) and his M.S. degree in Computer Science and Engineering (1999) from the University of Michigan. His research interests include data mining, data warehousing, and machine learning.

Toby J. Teorey is currently Professor of Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor. He received the B.S. (1964) and M.S. (1965) degrees in Electrical Engineering from the University of Arizona, Tucson, and a Ph.D. in Computer Science (1972) from the University of Wisconsin, Madison. He is the author of Database Modeling and Design (3rd edition, Morgan Kaufmann, 1999). Professor Teorey's current research interests include database design and data warehousing, OLAP, data mining, and advanced database systems. His internet address is teorey@eecs.umich.edu, and web home page is `http://www.eecs.umich.edu/~teorey`

References

[1] S. Agrawal, R. Agrawal, P. Deshpande, J. Naughton, S. Sarawagi, and R. Ramakrishnan. On the Computation of Multidimensional Aggregates. In *Proc. of the 22nd VLDB Conference*, 1996.

[2] E. Baralis, S. Paraboschi, and E. Teniente. Materialized View Selection in a Multidimensional Database. In *Proc. of the 23rd VLDB Conference*, 1997.

[3] D. Barbar, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey Data Reduction Report. *Data Engineering Bulletin 20(4)*, 1997.

[4] A. F. Cardenas. Analysis and Performance of Inverted Database Structures. *Comm. ACM*, 1975.

[5] C. Faloutsos, Y. Matias, and A. Silberschatz. Modeling skewed distributions using multifractals and the '80-20 law'. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, 1996.

[6] P. Flajolet and G.N. Martin. Probabilistic Counting Algorithms for Database Applications. *Journal of Computer and System Sciences*, 1985.

[7] M. Golfarelli and S. Rizzi. A Methodological Framework for Data Warehouse Design. In *Proc. ACM 1st Int. Workshop on Data Warehousing and OLAP*, 1998.

[8] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab and Sub-Totals. In *Proc. 12th ICDE*, 1996.

[9] H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proc. 6th ICDT*, 1997.

[10] H. Gupta, A. Harinarayan, A. Rajaraman, and J.D. Ullman. Index Selection for OLAP. In *Proc. 13th ICDE*, 1997.

[11] V. Harinarayan, A. Rajaraman, and J.D. Ullman. Implementing Data Cubes Efficiently. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data*, 1996.

[12] P. Hass, J.F. Naughton, S. Seshadri, and L. Stokes. Sampling-Based Estimation of the Number of Distinct Values of an Attribute. In *Proc. of the 21st VLDB Conference*, 1995.

[13] C.W. Chung J.H. Lee, D.H. Kim. Multi-Dimensional Selectivity Estimation Using Compressed Histogram Information. In *Proc. of the 1996 ACM-SIGMOD Conference*, 1999.

[14] V. Poosala and Y. E. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. In *Proc. of the 23rd VLDB Conference*, 1997.

[15] K. Runapongsa, H. Uchiyama, and T. Nadeau. Analysis of the Performance Parameter in ROLAP. `http://www.umich.edu/~krunapon/research/paper584.pdf`, Winter 1999. Term paper in EECS 584, Computer Science & Engineering, Univ. of Michigan.

[16] A. Shukla, P.M. Deshpande, and J.F. Naughton. Materialized View Selection for Multidimensional Datasets. In *Proc. of the 24th VLDB Conference*, 1998.

[17] A. Shukla, P.M. Deshpande, J.F. Naughton, and K. Ramasamy. Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. In *Proc. of the 22nd VLDB Conference*, 1996.

[18] A. Silberschatz, H.F. Korth, and S. Sudarshan. *Database System Concepts.* McGraw-Hill, 3rd edition, 1996.

[19] T.J. Teorey. *Database Modeling and Design.* Morgan Kaufmann Pub, 3rd edition, 1999.

[20] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd VLDB Conference*, 1997.

[21] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems.* Wiley, 1997.

[22] S.B. Yao. Approximating Block Accesses in Database Organizations. *Comm. ACM*, 1977.

[23] G.K. Zipf. *Human Behavior and Principle of Least Effort: an Introduction to Human Ecology.* Addison Wesley, Cambridge, 1949.