A Dissertation

entitled

Data Warehouse Operational Design: View Selection and Performance Simulation

by

Vikas R. Agrawal

Submitted as partial fulfillment of the requirements for
the Doctor of Philosophy degree in
Manufacturing Management and Engineering

Co-Advisor: Dr. Mesbah Ahmed

Co-Advisor: Dr. P. S. Sundararaghavan

Committee Member: Dr. Udayan Nandkeolyar

Committee Member: Dr. Robert Bennett

Graduate School

The University of Toledo

May 2005

An Abstract of

Data Warehouse Operational Design: View Selection and Performance Simulation

Vikas R. Agrawal

Submitted as partial fulfillment of the requirements for
the Doctor of Philosophy degree in
Manufacturing Management and Engineering

The University of Toledo

May 2005

Decision support systems are a key to gaining competitive advantage. Many corporations have built or are building unified decision-support databases called *data warehouses* on which decision makers can carry out their analysis. A data warehouse is a very large data base that integrates information extracted from multiple, independent, heterogeneous data sources to support business analysis activities and decision-making tasks. The data that is likely to be in demand is generally pre-computed and stored ahead of time at the data warehouse in the form of *materialized views*. This dramatically reduces execution time of decision support queries from hours or days to minutes or even seconds.

There are many architectural issues concerning the efficient design of a data warehouse. This dissertation studies in depth three important issues. The first issue addressed is the Materialized View Selection (MVS) problem, which is the problem of choosing an optimal set of views to materialize under resource constraints. We have formulated interesting bottleneck versions of this problem and presented the 0-1 Integer Programming models as well as the heuristic procedures. Performance analysis of the heuristic procedures is also presented.

Formulation of the MVS problem requires knowledge of the number of rows in each view in a given lattice structure, which refers to views and their interrelationships for a given set of dimensions. Counting actual number of rows present in each view takes considerable time. The second issue addressed in this dissertation focuses on the statistical sampling techniques applied to data warehouses to estimate number of rows in each view in a given lattice structure. We have shown that the application of sampling techniques results in significant time savings without compromising on accuracy.

The third issue deals with modeling the behavior and performance of a data warehouse system using simulation. We implemented the model in ARENA. The model enables a data warehouse manager to walk through various scenarios to investigate the synergy among various system components and to identify areas of inefficiencies in the system. This could also help improve overall performance of the data warehouse system.

ACKNOWLEDGEMENTS

I take this opportunity to thank my co-advisors Professor Mesbah Ahmed and Professor P. S. Sundararaghavan for their advice, support and encouragements during my pursuit of the Ph.D. program. They are the one who introduced me into the beauty of doing research in data warehousing. Their insights and expertise made my work easier and their enthusiasm towards research kept challenging me to seek the best quality of research.

I thank Professor Udayan Nandkeolyar and Professor Robert Bennett for their valuable suggestions and for serving as members of my doctoral advisory committee. I am fortunate to have their support during the program and I want to express my sincere thanks to them individually for their valuable advice and time.

I thank Professor T. S. Ragunathan, Professor William Doll and Professor Subba Rao for their help during my study at the University of Toledo.

I thank my friends and colleagues here at the University of Toledo, Shanawaz Muhammad, Ashish Thatte and Parag Dhumal who encouraged and helped me achieve this goal.

I thank my parents and my brother for their love and support. I thank my fiancé, Pragya, for her love and continuous support during this period with all my heart.

TABEL OF CONTENTS

LIST OF TABELS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

In today's fast-paced, ever-changing, wants-driven economy, information is seen as a key business resource to gain competitive advantage (Haag *et al.* 2005). To compete in today's turbulent market, businesses need to do considerable market research to find out what exactly people "want" rather than what they "need." The last three decades have seen an exponential growth in the area of information technology catering to the information processing needs of businesses in the form of capturing, storing, conveying, analyzing, and transferring data that will help knowledge workers and decision makers make sound business decisions.

With the widespread availability and ever-decreasing cost of computers, telecommunications technologies, and the Internet access, most of the businesses have collected a wealth of data. As a result, companies are becoming data rich though they remain information poor (Gray & Watson 1998, Grover 1998, Han & Kamber 2001). Valuable information gets lost in the shuffle and many companies struggle to get the right information, to the right person, and at the right time. This accentuates the need for turning the vast amount of data that is locked away in operational databases and other data sources into useful information that will help knowledge workers and decision makers gain access to the hidden knowledge and make the right decisions at the right time. One needs sound decision support systems to analyze this vast amount of data and to find solutions to business problems that are by nature often complex and unstructured.

This is exactly where *data warehousing* comes into the picture. With the competition mounting, most corporations have recognized that the data warehouse is a must-have strategic weapon to unearth the hidden business patterns, to recognize what exactly customers "want," and to understand the market competition.

Today, virtually all big corporations have built or are building unified data warehouses to support business analysis activities and decision-making tasks. The Palo Alto Management Group provides figures about the size and growth of the data warehousing market. They estimate that the sale of systems, software, services, and in-house expenditures will grow from a US $ 8.8 billion market in 1996 to a US $ 113.5 billion market in 2002 – a 51% compounded annual growth rate (Hillard *et al.* 1999, Walton *et al.* 2002).

In a typical organization, information is spread over many different multiple, independent, heterogeneous, and remote data sources. Acting as a decision support system, a data warehouse extracts, integrates and stores the "relevant" information from these data sources into one centralized data repository to support the information needs of knowledge workers and decision makers in the form of Online Analytical Processing (OLAP). Figure 1.1 depicts the general three-tier architecture for a typical data warehouse (Han & Kamber 2001).

The bottom tier is a warehouse database server that is almost always a relational database system. Data from the operational databases and other external sources are extracted using different application program interfaces.

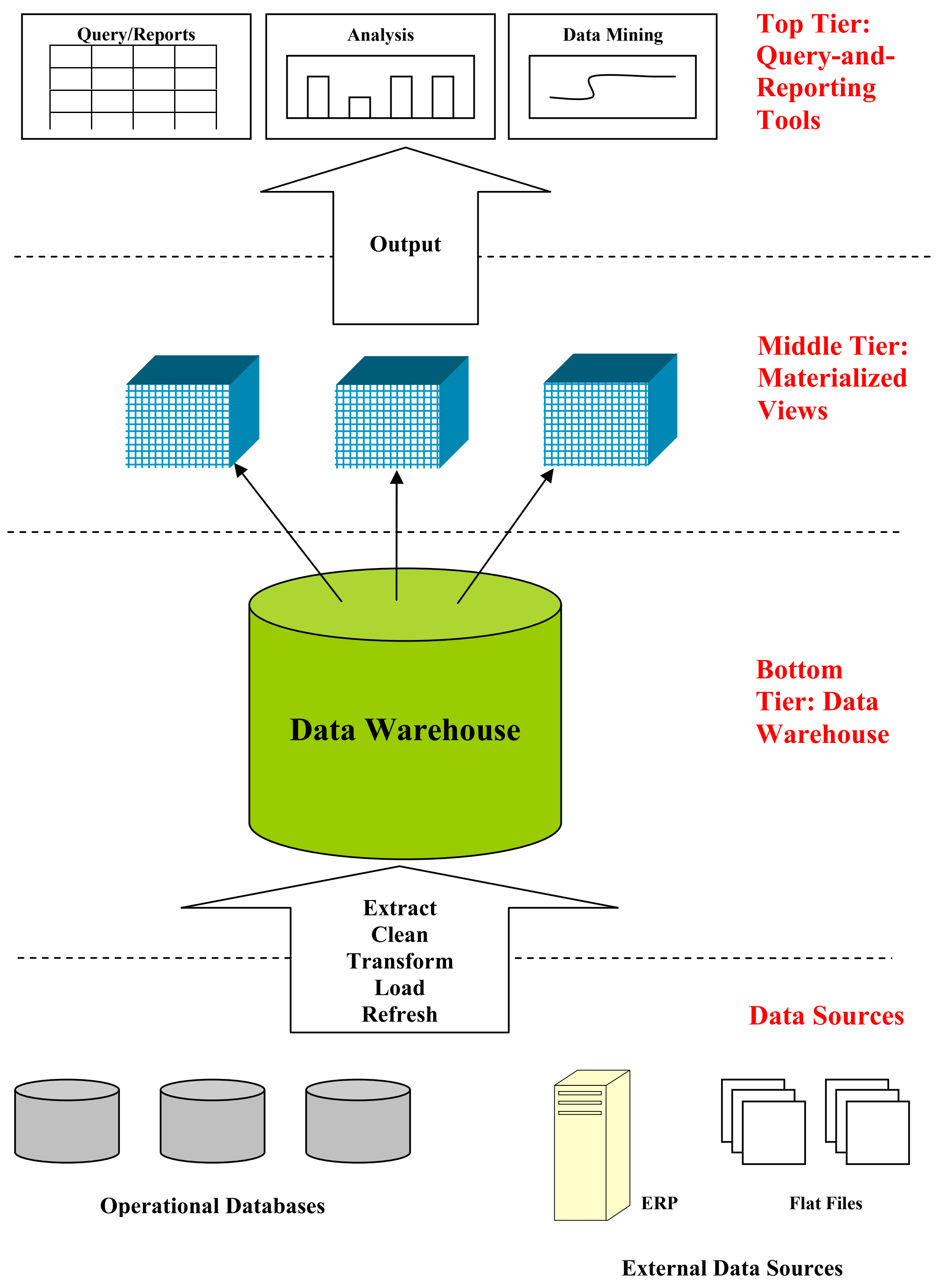

Figure 1.1: A Three-Tier Data Warehousing Architecture

The middle tier is the materialized views that store the aggregated and summarized data. This tier is typically implemented using either a relational OLAP (ROLAP) model or a multidimensional OLAP (MOLAP) model. The ROLAP is an extended relational DBMS that maps operations on multidimensional data to standard relational operations and the MOLAP is a special-purpose server that directly implements multidimensional data and operations. The top tier is a client interface, which contains query-and-reporting tools, multidimensional analysis tools, statistical analysis tools, and/or data mining tools.

Business analysts run business queries over this centralized data repository to gain the insights into the vast data and to mine for hidden knowledge. Results of such queries are generally pre-computed and stored ahead of time at the data warehouse in the form of *materialized views*. Such materialization of views reduces the query execution time to minutes or seconds which may otherwise take hours or even days to complete.

There are several architectural issues concerned with the design of a data warehouse. One such important design issue is to select the appropriate set of views to be materialized in the data warehouse. This is referred to as the Materialized View Selection (MVS) problem. We have formulated interesting bottleneck versions of this problem and proposed two heuristic procedures to solve the resulting MVS problem. We have also proposed exact solution techniques for reasonable sized problem instances.

The formulation requires data on the number of rows associated with each view in a given lattice structure. Querying to find the number of rows present in each view for a large lattice structure may not be practical as it takes considerable time. We have explored the use of sampling techniques to estimate the actual number of rows present in each view under consideration in a give lattice structure.

Data warehouses are considered to be large complex systems with many nonlinear interacting components. The performance of data warehouse systems is often unpredictable. Small changes in one part of the system could lead to large deterioration in performance. We have developed a simulation model to simulate the behavior and performance of a data warehouse system given its overall design.

In the remainder of this section, we briefly summarize the various problems addressed in this doctoral dissertation and present the contributions made.

1.1 Selection of Views to Materialize

Selecting the right set of views to be materialized is a non-trivial task. One cannot materialize all the views in a given lattice structure as such materialization is constrained by the availability of storage space, view maintenance cost, and computational time. On the other extreme, if one does not materialize any view then business queries have to be run over the source data, a process which would take considerable time. Such delay cannot be tolerated in today's decision support environment with its incredible demand on speed. Between these two extremes, one needs to find the optimum number of views to be materialized that will give reasonably good query response time while satisfying all the constraints. The MVS problem has been shown to be NP-Complete in the data warehouse literature (Harinarayan *et al*. 1999).

Given a set of queries to be supported, the conventional MVS problem is defined as follows: Given the resource constraints, select the set of views to be materialized so as to minimize the query response time. In this work, we have focused on the bottleneck version of the MVS problem, which changes the objective while leaving the rest unaltered. The objective is to *minimize the maximum weighted number of rows to be retrieved*, as opposed to the traditional objective of *minimizing the total number of rows to be retrieved*. We have also developed the 0-1 Integer Programming models for four different versions of the MVS problem and compared the results obtained from the heuristic procedures with that obtained from the 0-1 integer programming models.

1.2 Statistical Sampling to Instantiate MVS Problem Instances

Instantiation of an MVS problem for a given lattice structure requires the knowledge of the actual number of rows present in each view in a given lattice structure. Clearly this is needed for implementing the heuristic procedures or the exact solution techniques. This actually requires processing many complex queries to find the number of rows in all views in a given lattice structure. This would require considerable amount of time and such delays may be unacceptable in today's decision support environments where time plays a crucial role in the decision making process.

Hence, we have explored the use of statistical sampling techniques to address this issue. We have created two large realistic data warehouses and employed three estimators from the database sampling literature to estimate the number of rows present in each view in a given lattice structure using these two data warehouses. These estimators are the Shlosser Estimator (Shlosser 1981, Hass *et al.* 1995), the Guaranteed-Error Estimator (Charikar *et al.* 2000) and the Adaptive Estimator (Charikar *et al.* 2000). The next question is the sensitivity of the heuristics to the estimated instantiations as compared to the exact instantiations and their overall comparison to exact solutions. We addressed this by employing the 0-1 Integer Programming model to the exact instantiations and the heuristics developed by Harinarayan *et al.* (1999) to both the instantiations i.e. exact as well as estimated, and compared the results. Our findings suggest a good amount of savings in terms of computation time without compromising on accuracy.

1.3 Simulation Model and Analysis of a Data Warehouse

A data warehouse is a long term commitment for many organizations that promises high rewards, with potentially high risks. Data warehouses are large complex systems with many nonlinear interacting components. The data warehouse environment is considered to be very sensitive and dynamic. A small change in any one component may produce dramatically different changes somewhere else in the system. Often it is difficult for the data warehouse manager to predict the system's performance. Data warehouse managers need to experiment with the real system components to optimize the data warehouse design, which is an often tedious and risky endeavor.

We have developed a simulation model using the *ARENA* simulation package that will simulate the behavior and performance of a data warehouse system given its overall design. We had to overcome some challenges in building such a model which adequately represents a multitasking processor environment within the constraints posed by the model building tool. Given such a model, a data warehouse manager can walk through various what-if scenarios and can pinpoint the areas of inefficiencies in the system. This could result in improved data warehouse system performance.

1.4 Dissertation Organization

The rest of the dissertation is organized as follows. In Chapter 2, we address the problem of selecting the set of views to materialize in the data warehouse given two different resource constraints. Chapter 3 deals with statistical sampling techniques employed to formulate the MVS problem instances. In Chapter 4, we address the issues concerned with the simulation of the data warehouse system environment. We end with conclusions and future directions in Chapter 5.

CHAPTER 2: SELECTION OF VIEWS TO MATERIALIZE

2.1 Introduction

Decision support systems (DSS) are rapidly becoming a key to gaining competitive advantage for businesses. DSS allow businesses to get at the vast amount of data that is locked away in operational databases and other data sources and to turn that data into useful information. Many corporations have built or are building unified decision-support databases called *data warehouses* on which users can carry out their data analysis. A typical data warehouse extracts, integrates, and stores the *relevant information* from multiple, independent, and heterogeneous data sources into one centralized data repository to support the decision making information needs of knowledge workers and decision makers in the form of Online Analytical processing (OLAP) (Han & Kamber 2001, Harinarayan *et al.* 1999).

While operational databases maintain current information, data warehouses typically maintain information from a historical perspective. Hence, data warehouses tend to be very large and grow over time. Also, users of DSS are more interested in identifying hidden trends rather than looking at individual records in isolation. As a result, decision support queries are more complex than Online Transaction Processing (OLTP) queries and call for heavy use of aggregations.

The size of the data warehouse and the complexity of queries can cause decision support queries to take a very long time to complete. This delay is unacceptable in most DSS environments as it severely limits productivity. The usual requirement is that most query execution times be a few seconds or at most a few minutes (Gupta 1999, Harinarayan *et al*. 1996, 1999).

Many techniques have been discussed in the literature to improve the query response time performance goals. Query optimization and query evaluation techniques can be enhanced to handle the aggregations better (Chaudhuri & Shim 1994, Gupta *et al.* 1995). Also different indexing strategies like bit-mapped indexes and join indexes could be used to handle group-by(s) better (O'Neil & Graefe 1995).

One such commonly used technique is to materialize (pre-compute and store) the results of frequently asked queries. But picking the "right" set of queries to materialize is a nontrivial task. One may want to materialize a relatively infrequently asked query if it helps in answering many other frequent queries faster.

In this chapter, we formulate the 0-1 Integer Programming models that determine the optimal set of views to be materialized for four different versions of the MVS problem. Of these, two are bottleneck versions of the MVS problem. The objective in two of the four versions is to minimize the total weighted number of pages to be retrieved, and the objective in the other two versions is to minimize the maximum weighted number of pages to be retrieved. The former objective attempts to reduce the processing burden on the system while the later attempts to establish a ceiling on the amount of time a user will have to wait to obtain the required data. We also develop and present the heuristics for the bottleneck versions of the MVS problem.

The next section briefly discusses the conceptual background of the materialized view selection (MVS) problem. The MVS problem is defined and discussed briefly in Section 2.3. Section 2.4 discusses the 0-1 Integer Programming models for four different versions of the MVS problem. The heuristic procedures developed for the MVS problem are presented in Section 2.5. In Section 2.6, we present the performance of the 0-1 Integer Programming models and the heuristic procedures on typical problem sets. We end the chapter in Section 2.7 with the concluding remarks and future research directions.

2.2 Conceptual Background and Related Work

A data cube is a multi-dimensional modeling construct. For a detailed discussion of data cubes and cuboids the readers may refer to Han and Kamber (2001) and Microsoft SQLServer 2000 Analysis Services (Jacobson, R. 2000). A data cube contains many cuboids. A cuboid is also commonly known as a "view." In this context, a view is a set of aggregated data for a particular set of dimensions. Essentially, a view is the result of a "GROUP BY" query.

In a given data cube, the following implementation alternatives are possible:

1. Physically materialize the whole data cube. This is known as 100% materialization of a data cube. This approach will give the best possible query response time. Obviously, 100% materialization may be infeasible for a large data cube because it will require an excessive amount of disk space. Also, the time required to materialize a view is considerable. So 100% view materialization might take a long time to

accomplish, which might not be affordable in today's decision support environment. Also one needs to maintain indices, if any, which will further add to overall cost. Once views are materialized, they need to be maintained to reflect the current or the latest updates in the source data. Hence, as more views are materialized, the view maintenance costs will also increase.

2. Do not materialize any view. In this case, one needs to access the raw data and answer each query. This approach will result in long retrieval times due to high CPU and disk load. But it does not need any extra storage space for the view materialization.

3. The third alternative is to materialize only a part of the data cube. But selecting the right set of views to materialize is the challenge. In a data cube, many views could be derived from other views. Consequently, one may want to materialize a relatively infrequently accessed view if it helps in obtaining many other views quickly. We will refer to this problem as the Materialized View Selection (MVS) problem. The MVS problem has been shown to be NP-Complete problem (Harinarayan *et al*. 1999).

In the1980s, materialized views were investigated to speed up the data retrieval process for running queries on views in very large databases (Adiba & Lindsay 1980). Subsequently, further research studies were reported in view and index maintenance along with comparative evaluations of materialized views on the performance of queries (Blakeley & Martin 1990, Qian & Wiederhold 1991, Segev & Fang 1991).

Gray *et al.* (1997) proposed the data cube as a relational aggregation operator generalizing group-by, cross-tabs, and subtotals. Harinarayan, Rajaraman and Ullman (1996, 1999) have discussed the major features of the MVS problem elaborately. They have employed a lattice framework to capture the dependencies among views. This lattice framework was then used to develop a greedy algorithm (this will be referred to as the *HRU Heuristic 1* in the rest of the dissertation) for a special case of Problem 1 (refer Section 2.3). The details of the heuristic and the underlying assumptions are reproduced here for the case of reference.

Assumptions behind the HRU Heuristic 1

1. The cost of constructing a view from its materialized ancestor is a linear function of the number of rows in its materialized ancestor.
2. If view *i* is materialized, its storage cost will be r_i, where r_i is the number of rows in view *i*.
3. Whenever a user (or an application) requests a view, the request is always for the entire view and not for any part of it. For example, if someone requests the annual sales, he/she requests the annual sales for each year, and not for a specific year.
4. The views are either stored or created from relational database tables.

HRU Heuristic 1:

Consider a data-cube lattice with space costs associated with each view. Here the space cost is the number of rows in the view. Let $C(v)$ be the cost of view v. The set of views to be materialized should always include the top view (base cuboid), as there is no other view that can be used to answer the query corresponding to that view. Suppose there is a limit k on the number of views, in addition to the top view, to be selected for materialization. After selecting some set S of views, the *benefit* of view v relative to S, denoted by $B(v,S)$, is defined as follows.

1. For each view $w \leq v$ (i.e. w can be totally obtained from v), define the quantity B_w by:
    a. Let u be the view of least cost in S such that $w \leq u$. Note that since the top view is in S, there must be at least one such view in S.
    b. If $C(v) < C(u)$, then $B_w = C(v) - C(u)$. Otherwise, $B_w = 0$.
2. Define $B(v,S) = \sum_{w \leq v} B_w$.

Here, the benefit of v is computed by considering how it can improve the cost of evaluating views, including itself. For each view w that v covers, the cost of evaluating w using v is compared, and using whatever view from S offered the cheapest way of evaluating w. If v helps, i.e., if the cost of v is less than the cost of its competitor, then the difference represents part of the benefit of selecting v as a materialized view. The total benefit $B(v,S)$ is the sum over all views w of the benefit of using v to evaluate w, provided that benefit is positive.

The *Greedy Algorithm* for selecting a set of k views to materialize is given below:

```
$S$ = {top view};
for $i$ = 1 to $k$ do begin
        select that view $v$ not in $S$ such that $B(v,S)$ is maximized;
        $S$ = $S$ union {$v$};
end;
resulting $S$ is the greedy selection;
```

As seen from the above description, the HRU Heuristic 1 attempts to minimize the average time taken to evaluate the set of queries that are identical to the views in a given data cube. This algorithm first finds the benefit for each view in the data cube if it were to be materialized and then materializes the one with the highest benefit. It continues this process until the given number of views to be materialized constraint is satisfied.

Instead of asking for some fixed number of views to be materialized, one might instead allocate a fixed amount of storage space to views (other than the top view, which must always be materialized). Harinarayan *et al.* (1999) pointed out that in this case one needs to consider the benefit of each view *per unit space* used by a materialization of that view. All the assumptions underlined HRU Heuristic 1 applies in this case as well. One can easily modify HRU Heuristic 1 to address the storage space constraint. We will refer to this as *HRU Heuristic 2*, which is the heuristic for a special case of Problem 2 (refer Section 2.3) in the rest of dissertation. The stepwise procedure for HRU Heuristic 2 is outlined below:

HRU Heuristic 2:

Let *TS* is the total space available for materialization of views and let *SC* is the space consumed by the materialized views. Initially *SC* will be equal to the space consumed by the top view, which is always materialized.

The *Greedy Algorithm* for selecting a set of views to materialize while satisfying the storage space constraint is given below:

```
$S$ = {top view};
while $SC < TS$ begin
    select that view $v$ not in $S$ such that $B(v,S)/C(v)$* is maximized;
    $S = S$ union $\{v\}$;
    $SC = SC + C(v)$
end;
resulting $S$ is the greedy selection;
```

* please refer to *HRU Heuristic 1* for steps for calculating $B(v,S)$ and for $C(v)$

Harinarayan *et al.* (1999) have developed heuristics for the basic versions of the MVS problem i.e. Problem 1 with equal weights and Problem 2 with equal weights (refer to Section 2.3). In their paper they have identified a number of issues that required further investigation. Some of these major issues are:

1. The views in a lattice are unlikely to have the same probability of being requested in a query. Rather, one might be able to associate some probability (weight) with each view, representing the frequency with which it is queried and/or the importance of the person accessing the view.

2. In case of the storage space constraint, the greedy algorithm again seems appropriate, but there is additional complication that one might have a very small view with a very high benefit per unit space, and a very large view with almost the same benefit per unit space. Choosing the small view excludes the large view, because there is not enough space available for the large view after one chooses the small. However, they further pointed out that if one ignores “boundary cases”

like the one above, the performance guarantee of the greedy algorithm is the same as in the simple case.

3. HRU heuristics (HRU Heuristic 1 and HRU Heuristic 2) as well as the heuristics detailed in here (refer to Section 2.5) require prior knowledge of the number of rows present in each view in a given lattice structure. One could use sampling techniques to estimate the size of the other views by drawing a sample from the root view.

In this work, we have extended the work done by Harinarayan *et al.* (1999). We have provided extensive experimental evaluations by considering all of the above important issues. We have also developed two bottleneck versions of the MVS problem (refer to Section 2.5) that attempts to minimize the maximum weighted time taken to evaluate the set of queries that are identical to the views in a given data cube. We have also used three different sampling techniques (refer to Chapter 3) to estimate the size of all the other views by drawing samples from the root view.

In the next section, we define some variants of the MVS problem.

2.3 Materialized View Selection Problem

Figure 2.1 presents a hypothetical data cube and the associated cuboids in a hypothetical data warehouse. Each node represents a required view (cuboid) and the numbers inside each node represent the number of pages that must be retrieved to respond to the underlying query and the weight associated with each view, which is a function of frequency of access and/or the importance of the user accessing the view. We are using pages rather than rows as a surrogate for estimating the time it will take to obtain the required views. This is because most database software retrieve blocks of rows called pages during each physical access of the database. The page is then stored in cache or RAM from where rows can be retrieved quickly. Consequently, the number of pages is a better estimator of the time needed to obtain a required view.

View A (at the root) contains the lowest level of aggregated data, and it is assumed to be always materialized. The links in the lattice indicate parent-child relationships. Hence view B, for example, can be obtained from view A by processing 100 pages of data. If view B is materialized, it will contain 50 pages of data, and a query on view B will involve retrieving 50 pages. In general, for a given node, an ancestor node is defined as any node from which the given node may be reached by traversing only directed arrows. A query on a view may be answered by materializing the corresponding view or from any of its materialized ancestor views but not from any other materialized views. For example, obtaining view E from view A will require retrieving 100 pages while it will require retrieving 50 pages to obtain it from view B. View E cannot be obtained from view C or view D.

Based upon this type of configuration, we would like to know the specific set of views that must be materialized to achieve some predetermined objectives and constraints. We consider two types of objectives i.e. minimizing the total weighted number of pages to be retrieved, and minimizing the maximum weighted number of pages to be retrieved. We also consider two types of constraints – the maximum number of views that may be materialized and the total amount of storage space available to store the materialized views.

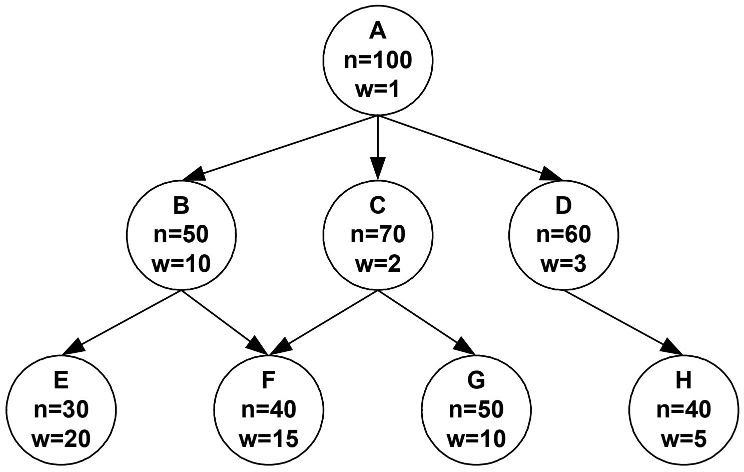

Figure 2.1: A Graphical Representation of a Data Cube with Associated Cuboids

Minimizing the total weighted number of pages to be retrieved is a commonly used objective. In practice this represents the amount of computing effort required to obtain all of the required views given either the frequency with which each view is accessed or the relative importance of the various views. If one assumes that all of the required views in a data cube are equally important, or that we do not have any information about the relative importance of the various required views in a data cube, we get a special case of this objective wherein all weights are equal.

Minimizing the maximum weighted number of pages to be retrieved attempts to limit the amount of time it will take to obtain any of the views. This is a bottleneck objective as it tries to minimize the maximum value. This measure also takes into account the relative importance of the various views. This objective will help improve the response time of the system.

Limiting the number of views that may be materialized is a commonly used constraint and attempts to limit the complexity of the data warehouse (Harinarayan *et al*. 1996, 1999). The storage of too many views will make the data warehouse more complex, increase the amount of time and effort required to both compute and maintain the various views. In addition, there may be a limit on the amount of space available to store the materialized views. Hence, from a practical point of view, a more realistic constraint might be to compute the storage requirement of the views and limit this to the amount of space available.

The discussions above lead us to define the following four problems:

Problem 1. Given a data cube, the maximum number of views that can be materialized, the weight associated with each required view, and the list of required views, determine the set of views to be materialized so as to minimize the weighted total number of pages to be retrieved in order to obtain each required view in the data cube. This reduces to the standard MVS problem with the maximum number of views to be materialized constraint when all weights are equal.

Problem 2. Given a data cube, the weight associated with each required view, the list of required views, and the maximum number of pages that can be stored, determine the set of views to be materialized so as to minimize the total weighted number of pages to be retrieved in order to obtain each of the required view in the data cube. Again, this reduces to the MVS problem with maximum number of pages constraint when all weights are equal.

Problem 3. Given a data cube, the maximum number of views that can be materialized, the weight associated with each required view, and the list of required views, determine the set of views to be materialized in order to minimize the maximum weighted number of pages to be retrieved so as to obtain each required view in the data cube.

Problem 4. Given a data cube, the weight associated with each required view, the list of required views, and the maximum number of pages that can be stored, determine the set of views to be materialized so as to minimize the maximum weighted number of pages to be retrieved in order to obtain each required view in the data cube.

In the next section, we present the 0-1 Integer Programming models for four different versions of the MVS problem discussed in this section.

2.4 Integer Programming Models for the MVS Problem

Below, we have presented Integer Programming models for each problem defined in the last section. Since the MVS problem is NP-complete (Harinarayan *et al*. 1999), it will not be practical to solve very large problems using these formulations.

Problem 1:

$$Min: \sum_i \sum_j w_j x_{ij} P_{ij} \tag{1}$$

Such that:

$$\sum_i x_{ij} = 1, \quad \forall j \tag{2}$$

$$x_{ij} \le x_{ii}, \quad \forall i \ne j \tag{3}$$

$$\sum_i x_{ii} \le T \tag{4}$$

$$x_{ij} = 1 \; or \; 0 \tag{5}$$

Where:

$N = \{1, 2, 3, \ldots, T_{total}\}$

$i, j \in N$

w_j = Weight assigned to view j

P_{ij} = Number of pages associated with view *i*, if view *i* is an ancestor of *j*
= *M* otherwise

T = Maximum number of views that can be materialized

T_{total} = Total number of required views in a given data cube

Explanation:

For a given materialization scheme, (1) gives the minimum total weighted number of pages to be retrieved to generate all the required views in a data cube. The constraints in (2) ensure that every required view can be obtained, and each is obtained from exactly one source. The constraints in (3) ensure that a view j can be obtained from some view i only and only if view i is materialized. The constraint in (4) ensures that a maximum of 'T' views will be materialized. The constraints in (5) make sure that queries requiring view j is answered by using the appropriate materialized view. If x_{ij} = 1, it implies that queries on view j is answered using view i and 0 otherwise. x_{ii} = 1 implies that view i is materialized and 0 implies that it is not materialized. A special case of this formulation is obtained when all the weights are equal, and without loss of generality are set equal to 1, i.e., $w_j = 1$.

Example 1:

For the data cube shown in Figure 2.1 (except that $w_j = 1$ for $j \in N$), and the associated required views described in it, the P_{ij} matrix is shown in Table 2.1. The matrix shows that all views can potentially be obtained from view A upon reading 100 pages. Furthermore, views E and F can be obtained from view B upon reading 50 pages if view

B is materialized, and so on. Let $T = 5$. Solve this problem using the formulation given in Problem 1.

	A	B	C	D	E	F	G	H
A	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
B	10000	**50**	10000	10000	**50**	**50**	10000	10000
C	10000	10000	**70**	10000	10000	**70**	**70**	10000
D	10000	10000	10000	**60**	10000	10000	10000	**60**
E	10000	10000	10000	10000	**30**	10000	10000	10000
F	10000	10000	10000	10000	10000	**40**	10000	10000
G	10000	10000	10000	10000	10000	10000	**50**	10000
H	10000	10000	10000	10000	10000	10000	10000	**40**

Table 2.1: The P_{ij} Matrix for Example 1

	A	B	C	D	E	F	G	H
A	**1**	**1**	0	**1**	0	0	0	0
B	0	0	0	0	0	0	0	0
C	0	0	**1**	0	0	0	**1**	0
D	0	0	0	0	0	0	0	0
E	0	0	0	0	**1**	0	0	0
F	0	0	0	0	0	**1**	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	**1**

Table 2.2: The X_{ij} Solution Matrix for Example 1

Example 1 was solved using MS Excel and obtained the solution shown in Table 2.2. The solution indicates that views A, C, E, F, and H should be chosen for materialization to minimize the total weighted number of pages to be retrieved. The objective function value for this solution is 490. In addition the solution specifies that queries based on views B, D and G should be answered respectively using views A, A and C which are the most economical views among all views materialized by the solution. This accounts for all of the required views in the data cube. It is important to note that alternate optimal solutions also exist. For example, a solution where views A, B, C, D, E are materialized with view F obtained from view B, view G obtained from view C, and view H obtained from view D also gives an objective function value of 490.

Problem 2:

$$Min: \sum_i \sum_j w_j x_{ij} P_{ij} \tag{6}$$

Such that:

$$\sum_i x_{ij} = 1, \quad \forall j \tag{7}$$

$$x_{ij} \le x_{ii}, \quad \forall i \ne j \tag{8}$$

$$\sum_i x_{ii} P_{ii} \le S \tag{9}$$

$$x_{ij} = 1 \ or \ 0 \tag{10}$$

Where:

$N = \{1, 2, 3, \ldots, T_{total}\}$

$i, j \in N$

w_j = Weight assigned to view *j*

P_{ij} = Number of pages associated with view *i*, if view *i* is an ancestor of *j*
= *M* otherwise

S = Maximum number of pages that can be stored

T_{total} = Total number of required views in a given data cube

Explanation:

For a given materialization scheme, (6) gives the minimum total weighted number of pages to be retrieved to obtain all the views in a data cube. The constraints in (7) ensure that every required view can be obtained, and each is obtained from exactly one source. The constraints in (8) ensure that a view *j* can be obtained from some view *i* only and only if view *i* is materialized. The constraint in (9) ensures that the number of pages that can be stored does not exceed '*S*'. The constraints in (10) make sure that queries requiring view *j* is answered by using the appropriate materialized view. If x_{ij} = 1, it implies that queries on view *j* is answered using view *i* and 0 otherwise. x_{ii} = 1 implies that view *i* is materialized and 0 implies that it is not materialized. A special case of this formulation arises when all the weights are equal, and without loss of generality are set equal to 1, i.e., $w_j = 1$.

Problem 3:

$$Min : Z \tag{11}$$

Such that:

$$\sum_i x_{ij} = 1, \quad \forall j \tag{12}$$

$$x_{ij} \le x_{ii}, \quad \forall i \ne j \tag{13}$$

$$\sum_i x_{ii} \leq T \tag{14}$$

$$w_j \sum_i x_{ij} P_{ij} = Z_j, \quad \forall j \tag{15}$$

$$Z \geq Z_j, \quad \forall j \tag{16}$$

$$x_{ij} = 1 \ or \ 0 \tag{17}$$

Where:

$N = \{1, 2, 3, \ldots, T_{total}\}$

$i, j \in N$

w_j = Weight assigned to view *j*

P_{ij} = Number of pages associated with view *i*, if view *i* is an ancestor of *j*
= *M* otherwise

T = Maximum number of views that can be materialized

T_{total} = Total number of required views in a given data cube

w_j = Weight of view *j*

Z_j = Weighted number of pages that must be retrieved to obtain view *j*

Z = Maximum weighted number of pages that must be retrieved to obtain any of the required views in a given data cube

Explanation:

The objective function in (11) minimizes the maximum weighted number of pages to be retrieved to obtain all the required views in the data cube. Equations in (12) ensure that every required view can be obtained, and each is obtained from exactly one source. This will in general be the most economical view. However, if there are alternate solutions, this view may be any of the views materialized by the solution. Equations in (13) ensure that view *j* can be obtained from view *i* only and only if view *i* is materialized. Equation (14) ensures that a maximum of '*T*' views will be materialized. Equations in (15) compute the weighted number of pages retrieved to obtain each view *j*. Equations in (16) ensure that the optimal *Z* is greater than equal to all weighted number of pages retrieved for all of the views. The constraints in (17) make sure that queries requiring view *j* is answered by using the appropriate materialized view. If x_{ij} = 1, it implies that queries on view *j* is answered using view *i* and 0 otherwise. x_{ii} = 1 implies that view *i* is materialized and 0 implies that it is not materialized.

Example 2:

For the data cube shown in Figure 2.1, and the associated required views described in it, the P_{ij} matrix is shown in Table 2.1. Let *T* = 5. Solve this example using the formulation given in Problem 3.

	A	B	C	D	E	F	G	H
A	**1**	0	**1**	**1**	0	0	0	**1**
B	0	**1**	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0
E	0	0	0	0	**1**	0	0	0
F	0	0	0	0	0	**1**	0	0
G	0	0	0	0	0	0	**1**	0
H	0	0	0	0	0	0	0	0

Table 2.3: The X_{ij} Solution Matrix for Example 2

Example 2 is solved using MS Excel and obtained the solution shown in Table 2.3. The solution indicates that views A, B, E, F, and G should be chosen for materialization to minimize the maximum weighted number of pages to be retrieved. The objective function value for this solution is 600. In addition it tells us that views C, D, and H should be obtained from view A. This accounts for all of the '*T*' views to be materialized. This problem typically has alternate optimal solutions.

Problem 4:

$$Min : Z \quad (18)$$

Such that:

$$\sum_i x_{ij} = 1, \quad \forall j \quad (19)$$

$$x_{ij} \le x_{ii}, \quad \forall i \ne j \quad (20)$$

$$\sum_i x_{ii} P_{ii} \le S \quad (21)$$

$$w_j \sum_i x_{ij} P_{ij} = Z_j, \quad \forall j \quad (22)$$

$$Z \ge Z_j, \quad \forall j \quad (23)$$

$$x_{ij} = 1 \; or \; 0 \quad (24)$$

Where:

$N = \{1, 2, 3, \ldots, T_{total}\}$

$i, j \in N$

P_{ij} = Number of pages associated with view *i*, if view *i* is an ancestor of *j*
= *M* otherwise

S = Maximum number of pages that can be stored

w_j = Weight of view j

Z_j = Weighted number of pages that must be retrieved to obtain view j

Z = Maximum weighted number of pages that must be retrieved to obtain any of the required views in a given data cube

T_{total} = Total number of required views in a given data cube

Explanation:

The objective function in (18) requires the minimization of the maximum weighted number of pages to be retrieved to obtain all the views in the data cube. Equations in (19) ensure that every required view can be obtained, and each is obtained from exactly one source. This will in general be the most economical view. However, if there are alternate solutions, this view may be any of the views materialized by the solution. Equations in (20) ensure that view j can be created from view i only when view i is materialized. Equation (21) ensures that the number of pages that can be stored does not exceed 'S'. Equations in (22) compute the weighted number of pages retrieved to obtain each view. Equations in (23) ensure that the optimal Z is greater than equal to all weighted number of pages retrieved for all of the views. The constraints in (24) make sure that queries requiring view j is answered by using the appropriate materialized view. If $x_{ij} = 1$, it implies that queries on view j is answered using view i and 0 otherwise. $x_{ii} = 1$ implies that view i is materialized and 0 implies that it is not materialized.

In the next section, we present the heuristic procedures developed for various versions of the MVS problem.

2.5 *Heuristic Procedures for the MVS Problem*

In this section, we present the heuristic procedures for Problem 3 and Problem 4 defined in Section 2.3.

2.5.1 Selection of Views Under the Number of Views to be Materialized Constraint

We have employed the lattice framework (as used by the HRU heuristic) to capture the dependencies among the views to formulate our heuristic procedures for various versions the MVS problem. Below, we present a heuristic procedure for Problem 3 defined in Section 2.3:

Bottleneck Heuristic 1:

Step 1. Let k be the maximum number of views that may be materialized. Let $|N|$ denote the cardinality of the set N. Let set N be the set of all views under consideration, which will initially be all views except the root view A. Let M be the set of views to be materialized. Initially, let $M = \{A\}$ where view A is the root view which is required to be materialized. Let w_j be the weight associated with view j and let n_j be the number of pages required to represent materialization of view j. For each view j, calculate $f_j = w_j$ * (minimum number of pages to be retrieved to answer queries related to view j in the current solution which has materialized all views in *M*). Let the objective function value Z = Max (f_j for $j \in N \cup M$).

Step 2. For each view $j \in N$, calculate Z_j for $j \in N$, where Z_j is the objective function value if view *j* were to be materialized in addition to all the views in *M*. Let view $j' \in N$ be such that it maximizes (Z- Z_j) for $j \in N$ and (Z- Z_j) >0. If there is no such j', go to *Step 3*. Otherwise, let $M = M \cup j'$, $N = N - j'$. Set $Z = Z_{j'}$. If $|M| = k$ go to Step 3 else repeat *Step 2*.

Step 3. Views to be materialized are given by the set *M*. *Z* gives the objective function.

<u>Example 3:</u>

Applying Algorithm 1 to the problem whose lattice diagram is shown in Figure 2.1, we get the following:

Step 1. Let $M = \{A\}$ and $N = \{B, C, D, E, F, G, H\}$. The set of weights associated with views *A* through *H* be {1, 10, 2, 3, 20, 15, 10, 5}. Initial value of the objective function Z = Max{1*100, 10*100, 2*100, 3*100, 20*100, 15*100, 10*100, 5*100} = 2000.

Step 2. For each node *j* calculate Z_j. Next, details of the calculation of Z_B corresponding to materializing view B is given by Z_B = Max{1*100, 10*50, 2*100,3*100, 20*50, 15*50, 10*100,5*100} = 1000. Similarly, Z_j for nodes *C* through *H* is given by {2000, 2000, 1500, 2000, 2000, 2000}. So j' will be *B*. So $M = \{A, B\}$. $N = \{C, D, E, F, G, H\}$. $Z = 1000$.

Repeat *Step 2*. Calculate Z_j for nodes C through H = {1000, 1000, 1000, 1000, 1000, 1000}. Since there is no such node j that satisfies $(Z- Z_j) >0$ for $j \in N$, we go to *Step 3*.

Step 3. Applying *Step 3*, the solution is: $M = \{A, B\}$, and $Z = 1000$.

2.5.2 Selection of Views Under the Storage Space Constraint

Below, we present the stepwise procedure for Bottleneck Heuristic 2 for Problem 4 defined in Section 2.3:

Bottleneck Heuristic 2:

Step 1. Let S be the total storage space available before materializing the root view. Let set N be the set of all views under consideration, which will initially be all views except the root view A. Let M be the set of views to be materialized. Initially, let $M= \{A\}$ where view A is the root view which is required to be materialized and this will be the current solution. Let Q be the set of views that may not be considered. Initially Q will be empty. Let w_j be the weight associated with view j and let n_j, S_j respectively be the number of pages required, and space required to materialize view j. Let S_M be the space required to materialize all the views in the set M. For each node j, calculate $f_j = w_j$ *

(min. # or rows to be retrieved to answer queries related to view *j* in the current solution). Let the objective function value Z = Max (f_j for $j \in N \cup M$).

Step 2. For each view $j \in N - Q$, calculate Z_j for $j \in N - Q$ where Z_j is the objective function value if view *j* were to be materialized in addition to all other views in *M*. Let view $j' \in N - Q$ be such that it maximizes *(Z- Z_j)* for $j \in N - Q$ and *(Z- Z_j)* >0. If there is no such j', go to *Step 4* else go to *Step 3*.

Step 3. If there is such a j' and if $S - S_M - S_{j'} \geq 0$, then let $M = M \cup j'$, $N = N - j'$. Set $Z = Z_{j'}$. Go to Step 2. In case there are ties between more than one such j', they are broken arbitrarily. Else if $S - S_M - S_{j'} < 0$, that implies that there is not enough space left to accommodate view j'. Reset $Q = Q \cup j'$. Go to Step 2.

Step 4. Views to be materialized are given by the set *M*. *Z* gives the objective function.

In the next section, we discuss and compare the results obtained using HRU heuristics and Bottleneck heuristics (Bottleneck Heuristic 1 and Bottleneck Heuristic 2) with the results obtained using the 0-1 Integer Programming models.

2.6 Experimental Results

We randomly generated 10 instances of data cubes each with 32 required views and 64 required views. The procedure of generating problem instances is described in brief below.

For our experiments, we have generated problem instances for a given number of dimensions. We have not used any levels in those dimensions. For a given number of dimensions with no levels present in them, the number of cuboids present in a given data cube will be given by 2^n, where *n* is the number of dimensions.

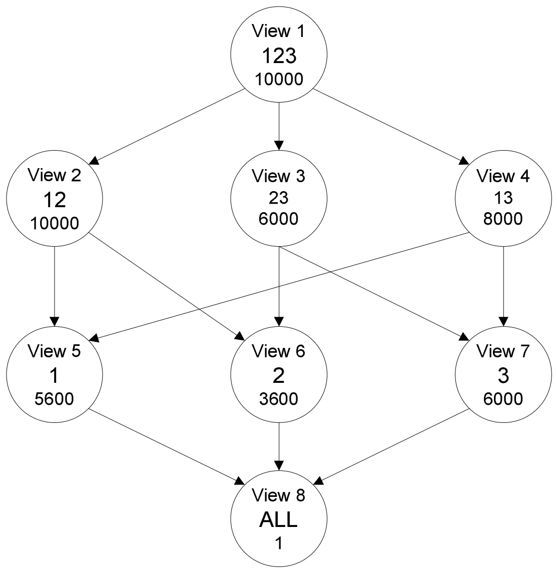

Figure 2.2: Data Cube Problem Instance

For example, let's say *n* = 3. So we will be having eight cuboids. Following diagram denotes the data cube and the associated cuboids.

We will name the root view as View 1 having data aggregated by three dimensions i.e. 1, 2 and 3, and assume that it will consist of 10000 rows. Next we will find all the possible combinations of two dimensions out of these three. So the possible combinations are View 2 having data aggregated by dimensions 1 and 2, View 3 having data aggregated by dimensions 2 and 3 and View 4 having data aggregated by 1 and 3 and all of them are answered from the root view 123. For each of these views, we will assume the number of rows will be drawn from a uniform distribution UNFI(0.6,1)*Min(number of rows in the set of immediate parents of the node). In this case, it has only one parent and it is the root view with 10,000 rows. Number of rows present in each of these views is shown in Figure 1.2. Next we will find all the possible combination of one dimension out of the three, which will result in the View 5, View 6 and View 7. Again in each of these views, the number of rows is generated as explained above. The last view will be having just one row which will be having data summarization by all the dimensions.

Next, we generated the 0-1 Integer Programming model for each of the problems defined in Section 2.3 and found the cost of optimal solution for each instance of these problems using LINGO software. Finally, we found the cost of solution to each of these problem instances using HRU Heuristics 1 and 2 (refer to Section 2.2), and the Bottleneck Heuristics 1 and 2 (refer to Section 2.5).

In Table 2.4, we have presented the cost comparison between the optimal solution, which is the number of pages to be retrieved, for the special case of Problem 1 described in Section 2.3, and the corresponding cost of solution obtained by the HRU Heuristic 1 under the number of views to be materialized constraint. Here, we have limited the number of views to be materialized to 10 and 20 for problems with 32 required views and 64 required views respectively. We found that the HRU Heuristic 1 comes within 1% of the optimal solution in all of the 20 problem instances. Furthermore, the HRU Heuristic 1 found the optimal solution for 10 of the 20 problem instances. For the size and complexity of problems tested here, the HRU Heuristic 1 seems to be a good method for solving Problem 1.

Problem Instance Number	Number of Required Views	Number of Views to be Materialized	Cost of Solution (no. of Pages to be retrieved)		% Deviation
			Optimal	HRU Heuristic 1	
1.1	32	10	163400	163500	0.06
1.2	32	10	153200	154250	0.69
1.3	32	10	131900	132000	0.08
1.4	32	10	138850	138850	0.00
1.5	32	10	164400	164400	0.00
1.6	32	10	171400	173000	0.93
1.7	32	10	176500	176640	0.08
1.8	32	10	182200	182200	0.00
1.9	32	10	159600	159600	0.00
1.10	32	10	134900	134900	0.00
1.11	64	20	262070	262070	0.00
1.12	64	20	224625	225715	0.49
1.13	64	20	248900	248900	0.00
1.14	64	20	232900	232900	0.00
1.15	64	20	250600	250600	0.00
1.16	64	20	264850	264990	0.05
1.17	64	20	303455	303455	0.00
1.18	64	20	230320	231600	0.56
1.19	64	20	269720	270440	0.27
1.20	64	20	305715	305995	0.09

Table 2.4: Cost Comparison Between Optimal Solution and HRU Heuristic 1 Solution for the Special Case of Problem 1

Problem Instance Number	Number of Required Views	Available Space as Proportion of Total Space	Cost of Solution (no. of Pages to be retrieved)		% Deviation
			Optimal	HRU Heuristic 2	
2.1	32	0.5	150431	151491	0.70
2.2	32	0.5	141811	142021	0.15
2.3	32	0.5	122016	122251	0.19
2.4	32	0.5	126451	126651	0.16
2.5	32	0.5	154481	154706	0.15
2.6	32	0.5	158201	158641	0.28
2.7	32	0.5	160801	160801	0.00
2.8	32	0.5	167741	167741	0.00
2.9	32	0.5	147701	147701	0.00
2.10	32	0.5	126101	126351	0.20
2.11	64	0.5	246025	246169	0.06
2.12	64	0.5	208958	208969	0.01
2.13	64	0.5	232896	233194	0.13
2.14	64	0.5	217278	217486	0.10
2.15	64	0.5	234274	234797	0.22
2.16	64	0.5	244027	244036	0.00
2.17	64	0.5	279617	280481	0.31
2.18	64	0.5	209525	210121	0.28
2.19	64	0.5	247951	248126	0.07
2.20	64	0.5	287368	288394	0.36

Table 2.5: Cost Comparison Between Optimal Solution and HRU Heuristic 2 Solution for the Special Case of Problem 2

Problem Instance Number	Number of Required Views	Number of Views to be Materialized	Cost of Solution (weighted number of pages to be retrieved)		% Deviation
			Optimal	Bottleneck Heuristic 1	
3.1	32	10	810000	810000	0.00
3.2	32	10	930000	930000	0.00
3.3	32	10	768000	768000	0.00
3.4	32	10	648000	648000	0.00
3.5	32	10	855000	855000	0.00
3.6	32	10	910000	910000	0.00
3.7	32	10	770000	770000	0.00
3.8	32	10	400500	436500	8.99
3.9	32	10	950000	950000	0.00
3.10	32	10	680000	680000	0.00
3.11	64	20	810000	810000	0.00
3.12	64	20	810000	980000	20.99
3.13	64	20	830000	830000	0.00
3.14	64	20	970000	970000	0.00
3.15	64	20	846000	980000	15.84
3.16	64	20	570000	570000	0.00
3.17	64	20	651000	960000	47.47
3.18	64	20	950000	950000	0.00
3.19	64	20	930000	930000	0.00
3.20	64	20	980000	980000	0.00

Table 2.6: Cost Comparison Between Optimal solution and Bottleneck Heuristic 1 Solution for Problem 3

Problem Instance Number	Number of Required Views	Available Space as Proportion of Total Space	Cost of Solution (weighted number of pages to be retrieved)		% Deviation
			Optimal	Bottleneck Heuristic 2	
4.1	32	0.5	810000	810000	0.00
4.2	32	0.5	930000	930000	0.00
4.3	32	0.5	768000	768000	0.00
4.4	32	0.5	648000	648000	0.00
4.5	32	0.5	855000	855000	0.00
4.6	32	0.5	910000	910000	0.00
4.7	32	0.5	770000	770000	0.00
4.8	32	0.5	400500	400500	0.00
4.9	32	0.5	950000	950000	0.00
4.10	32	0.5	680000	880000	29.41
4.11	64	0.5	810000	810000	0.00
4.12	64	0.5	810000	980000	20.99
4.13	64	0.5	830000	960000	15.66
4.14	64	0.5	970000	970000	0.00
4.15	64	0.5	846000	980000	15.84
4.16	64	0.5	570000	950000	66.67
4.17	64	0.5	651000	960000	47.47
4.18	64	0.5	950000	950000	0.00
4.19	64	0.5	930000	930000	0.00
4.20	64	0.5	980000	980000	0.00

Table 2.7: Cost Comparison Between Optimal solution and Bottleneck Heuristic 2 Solution for Problem 4

However, it is not possible to generalize this observation for situations where there are more required views and perhaps more complex dependencies. Harinarayan *et al.* (1999) have identified problem structures where their heuristic will not perform well. Furthermore, they have found an upper bound to the extent of the error. For situations requiring three views to be materialized (including the base cuboid), this upper bound is 25%.

The cost comparison between the optimal solution and the corresponding cost of solution obtained by the HRU Heuristic 2 under the storage space constraint for the special case of Problem 2 (refer to Section 2.3) is shown in Table 2.5. The amount of space available is assumed to be 50% of the total space required to materialize all the required views including the root view in the given data cube. We found that the HRU Heuristic 2 again comes within 1% of the optimal solution. However, in this case the HRU Heuristic 2 found the optimal solution only in 20% of the problem instances as compared to 50% of the problem instances in Problem 1 using HRU Heuristic 1.

Table 2.6 shows the cost comparison between the optimal solution and the cost of solution obtained by the Bottleneck Heuristic 1 under the number of views to be materialized constraint for Problem 3 (refer to Section 2.3). We found that the Bottleneck Heuristic 1 reached the optimal solution in all but four instances. In these instances the deviation varied between 8.99% and 47.47%. This is probably due to the heuristic's premature stopping condition as implemented in our computer program.

The cost comparison between the optimal solution and the cost of solution obtained by the Bottleneck Heuristic 2 under the storage space constraint for Problem 4 (refer to Section 2.3) appears in Table 2.7. We found that the heuristic did not find the optimal solution in six instances. In these instances the deviation from the optimal varied between 15.66% and 66.67%. Furthermore, our experimental evaluation of Problem 1 through Problem 4 points to the fact that the space constrained environment is more demanding on the heuristic; i.e., it seems to find the optimal solution in fewer instances.

In the next section, we conclude this chapter and suggest future research directions.

2.7 Conclusions and Future Research

In this chapter, we have presented two heuristic procedures for solving two versions of the MVS problem. In the first heuristic procedure (Bottleneck Heuristic 1), the constraint is the maximum number of views that can be materialized. In the second heuristic (Bottleneck Heuristic 2), the constraint is the total storage space available for materialization of views. We have also developed the 0-1 Integer Programming models for four different versions of the MVS problem. We used these formulations to solve 10 problem instances each with 32-node and 64-node data cubes. We then compared the cost of optimal solution with the corresponding cost of solution obtained by applying various heuristic procedures.

Our findings indicate that the heuristics used to solve the problem instances come very close to the optimal solutions. Even though the problem is NP-complete, we could solve 64 node problems within 10 seconds for Problem 1 and Problem 3, and within 90 seconds for Problems 2 and Problem 4 on a 466 MHz machine (refer to Section 2.3) using our formulation. It takes more time to solve problems with the storage space constraint as it puts higher burden on the system to check each possible combination to find the best set of views to materialize for obtaining optimal solution.

Further research in this area should perhaps be focused upon some of the related practical aspects of data warehouse management. For example, how well do the problems defined here address the real life problems faced by data warehouse administrators? The formulations presented here do not consider view maintenance costs. Further research is needed in this area. In the heuristic procedures developed here, weight for each node in a given lattice structure is generated randomly. In further research, one can also pursue the issues concerning developing a systematic process for defining the weights for views and performing sensitivity analysis of weights on the heuristics solutions.

CHAPTER 3: STATISTICAL SAMPLING TO INSTANTIATE MVS PROBLEM INSTANCES

3.1 Introduction

As described in the previous chapter, materialized views can speed up the execution of many queries (Gupta 1999, Harinarayan *et al.* 1999). For complex queries involving large volumes of data, the scope for speeding up using materialized views is very high. The HRU heuristics (HRU Heuristic 1 and HRU Heuristic 2) as well as the heuristic procedures reported in Chapter 2 assume that the number of pages (surrogate for rows) present in each node in a given lattice structure is known prior to applying the heuristic. In a large multidimensional data cube, there may be a large number of nodes in a lattice structure. Determining the number of rows in each of these nodes (views) may also be a time-consuming process. In these days of on-line real-time decision making, the data warehouse administrator may not be able to afford the necessary time of this magnitude before he/she applies a heuristic to determine the views to be materialized.

In this chapter, we explore the use of statistical sampling methodologies to estimate the number of rows present in each node in a given lattice structure. Here, we estimate the rows, which is a surrogate measure for the pages used in previous chapter. We assume that the root node is always materialized as were the cases in Harinarayan, Rajaraman, and Ullman's work (1996, 1999). The root node is assumed to contain the lowest level of aggregated data and the keys to appropriate levels of the dimensions of a cube. Thus, it is possible to take a sample from it and estimate the number of rows that is expected to be in a particular derived node.

In the database related literature, several researchers have developed a number of sampling techniques to estimate the number of unique rows in a relation. We employ three of these sampling techniques to estimate the number of rows present in each node of a 27-node lattice structure. Subsequently we apply appropriate HRU heuristic (HRU Heuristic 1 or HRU Heuristic 2) to solve the MVS problem (Problem 1 with equal weights or Problem 2 with equal weights, respectively) twice: once with the actual number of rows, and once with the estimated number of rows. We also apply the 0-1 Integer Programming model to solve Problem 1 with equal weights, and actual number of rows to determine the optimal solution. Finally we compare the solutions obtained by applying the HRU heuristics on the actual as well as the estimated data with the optimal solution obtained by applying the 0-1 Integer Programming model on the actual data set.

In the next section, we present the sampling literature related to finding the number of unique rows in a relation. In Section 3.3, we present the methodology adopted for estimating the number of rows for the MVS problem instances for two realistic data warehouses. In Section 3.4, we discuss two realistic lattice instances that we generated for testing purposes, and present the results of our experiments. The overall results are briefly discussed in Section 3.5. We conclude the chapter with possible future research directions in Section 3.6.

3.2 Sampling Literature Related to the Number of Unique Rows

Suppose that the root node *View0* contains the monthly sales data for different cities in different countries. Also, suppose that the SQL query for a descendant view *View1* is "SELECT SUM (sales) FROM *View0* GROUP BY country." If *View0* contains *n* unique countries, it is clear that *View1* will also contain exactly *n* rows. Thus, estimating the number of rows in a view of a data cube is essentially the same as estimating the number of distinct values of an attribute in a relation. Since the root view contains the least aggregated data, and since it is always materialized, it can be used to estimate the number of rows in all other views.

In the statistics literature, the problem of estimating the number of classes is a well-studied problem that is equivalent to the distinct-value estimation problem in the data warehouses (Bunge and Fitzpartick 1993). This problem has also been extensively researched in the database literature. For a thorough analysis refer to the work by Hass *et al.* (1995). They have devised several new estimators including the *hybrid estimator* that appears to outperform the estimators developed in prior literature, both for real and synthetic data set. This *hybrid estimator* first uses a chi-squared test to decide whether the data has low skew or high skew. Accordingly, it applies a *smoothed jackknife* estimator in the former case and the *Shlosser* Estimator in the latter case. In this dissertation, we will denote this as HYBSHLO estimator. Hass and Stokes (1998) have done an extensive study of several generalized jackknife estimators, relating previously known estimators and proposing new ones.

Charikar *et al.* (2000) have devised an estimator called Guaranteed-Error Estimator (GEE). They pointed out that the GEE performed well for data with high skew or with relative few low frequency elements, which was demonstrated in their experimental results. By the same reasoning, they pointed out that the GEE would not perform as well (in fact will severely underestimate) for data which has *both* low skew and a large number of distinct values; again, this was demonstrated in their experimental results. In case of high-skew real-world data as well as synthetic data, they found that the GEE outperforms the Shlosser Estimator. So they suggested a modified version of HYBSHLO estimator which substitutes the GEE for the Shlosser estimator in the case of high-skew data. This is denoted as the HYBGEE (for Hybrid with the GEE). Their experimental results show that HYBGEE gives a significant reduction in error compared to HYBSHLO.

Charikar *et al.* (2000) through their extensive empirical evaluation, further pointed out that to ensure accuracy, the estimation procedure needs to take into account characteristics of the input distribution. Keeping this in mind, they analytically derived an estimator called the Adaptive Estimator (AE), a modified version of the GEE that adapts to the input distribution of data so as to avoid the problem faced by the GEE for data having both low skew and large number of distinct values.

In this dissertation, we have investigated three estimators. These are the Shlosser Estimator (Shlosser 1981, Hass *et al.* 1995), the GEE (Charikar *et al.* 2000), and the AE (Charikar *et al.* 2000). The details of these estimators are available in the respective referenced papers. In this section, we will simply include the computational expressions that have been used to estimate these three statistics.

Let,

n = number of rows in a relation,

r = number of rows in a sample drawn from a relation,

d = number of distinct values of an attribute or of a composite attribute that appears in the sample

$\hat{D}$ = estimate for the number of distinct values of an attribute or a composite attribute that exists in the population.

For $1 \le i \le r$, let f_i be the number of attribute values that appear exactly i times in the sample of size r. Thus, $d = \sum_{i=1}^{r} f_i$ and $r = \sum_{i=1}^{r} if_i$

We applied sanity bounds to all the estimators used to ensure that $d \le \hat{D} \le n$; i.e., if $\hat{D} > n$, we set $\hat{D}$ to n and if $\hat{D} < d$, we set $\hat{D}$ to d (Charikar *et al.* 2000). With the above notations, we have computed the relevant estimates using the following computational expressions.

Shlosser Estimator: (Ref: Hass *et al.* 1995, and Shlosser 1981)

- $$\hat{D}_{Shloss} = d + \frac{f_1 \sum_{i=1}^{r} (1-q)^i f_i}{\sum_{i=1}^{r} iq(1-q)^{i-1} f_i}$$

where, $q = r/n$ (is the probability with which each tuple is included in the sample, independently of all other tuples.)

<u>GEE: The Guaranteed-Error Estimator:</u> <u>(Ref: Charikar *et al.* 2000)</u>

- $\hat{D}_{GEE} = (\sqrt{\frac{n}{r}}) f_1 + \sum_{i=2}^{r} f_i$

<u>AE (The Adaptive Estimator):</u> <u>(Ref: Charikar *et al.* 2000)</u>

- $m - f_1 - f_2 = f_1 \dfrac{\sum_{i=3}^{r} e^{-i} f_i + m e^{-(f_1+2f_2)/m}}{\sum_{i=3}^{r} i e^{-i} f_i + (f_1 + 2f_2) e^{-(f_1+2f_2)/m}}$
- $\hat{D}_{AE} = d + m - f_1 - f_2$

where m is the number of low frequency values.

In the next section, we briefly outline the methodology adopted in our experiments.

3.3 Methodology

Consider a lattice with n views named $v_0, v_1, v_2 \ldots v_{n-1}$. Let r_i and $\hat{r}_i$ be the actual and estimated number of rows in v_i, respectively. We can employ the HRU heuristics (HRU Heuristic 1 and HRU Heuristic 2) to solve Problem 1 with equal weights or Problem 2 with equal weights (refer Chapter 2), as warranted, using the values of r_i's. Our objective is to investigate if we can use the estimated values of the r_i's and get an acceptable solution. To accomplish the objective, we have employed the following procedures:

1. Use the HRU heuristics to solve an instance of Problem 1 with equal weights or Problem 2 with equal weights, as warranted, using the actual number of rows in the views.
2. Employ three alternative estimators to estimate the number of rows in the views. Apply these estimates to solve the problem instances generated using each estimating method.
3. Use the 0-1 Integer Programming model to determine the optimal solution (with the actual number of rows in each view).
4. Finally compare the above solutions.

For generating problem instances, we need a lattice structure. For our investigation, we have constructed two 27-node lattice instances viz. Lattice Instance 1 (TPCH database) and Lattice Instance 2 (AANS database). These lattice instances are discussed briefly below.

3.3.1 Lattice Instance 1 (TPCH Database)

In this case, we have first populated a 1-GB TPCH Benchmark database (http://www.tpc.org/tpch/spec/h130.pdf). Then we populated the root node from this database using the Customer (C), Part (P), Month (M) dimensions, and the "Sale" measure.

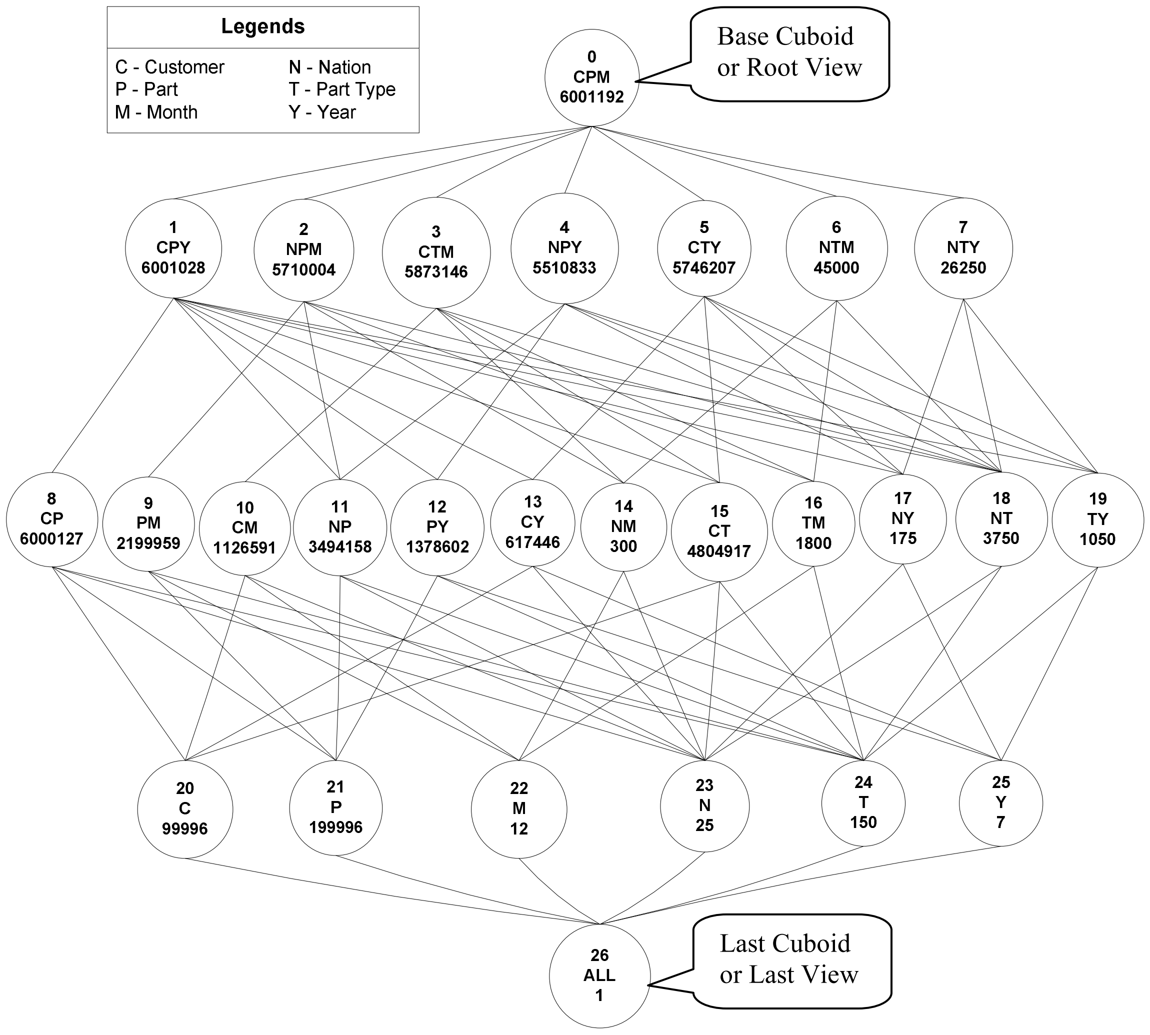

Figure 3.1: Lattice Instance 1 with Actual Number of Rows
(Source: TPCH Benchmark Database)*
* To keep the diagram simple, many dependencies have not been shown

The actual number of rows in each view was counted from the root node. In the remainder of the chapter we will refer to this database as the TPCH database. With three dimensions and two levels in each dimension, we have a 27-node lattice instance. This lattice diagram along with the row counts is shown in Figure 3.1.

3.3.2 Lattice Instance 2 (AANS Database)

At 1-GB level, the TPCH database does not represent a realistic transaction database. For example, no customer appears to have ordered the same product more than once. Thus some of the higher level nodes tend to have the same number of records as do their parents (see Figure 3.1). This is not a shortcoming of the TPCH database. We assume that if the root node were instantiated from a very large database (like 100 GB), perhaps it would have behaved differently. However, the limitations on our hardware/software configuration forced us not to go beyond a 1-GB database. This is why we created our own 1-GB database. In the remainder of the chapter we will refer to this database as the AANS database. In this database, the daily transactions were created randomly from certain predefined probability distributions with the expectation that the number of rows in a descendent view would be in general be somewhat less than the number of rows in its ancestor view. Again we have considered the same three dimensions as were the case in TPCH database. This 27-node instance of the lattice along with row counts is shown in Figure 3.2

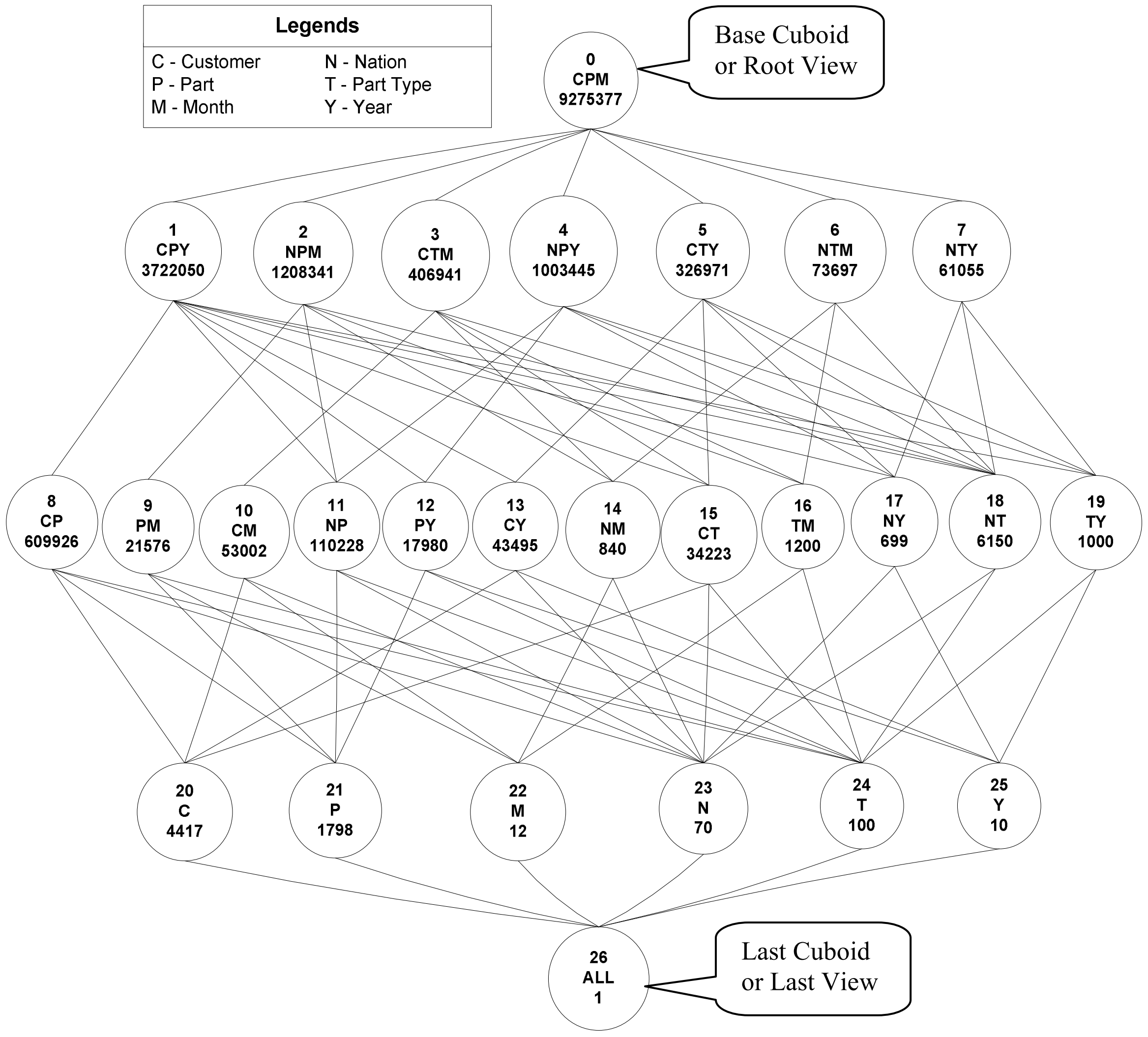

Figure 3.2: Lattice Instance 2 with Actual Number of Rows
(source: AANS Database) *
* To keep the diagram simple, many dependencies have not been shown

3.3.3 Problem Instances for Experimentation

Table 3.1 briefly defines the experimental setup that we have used in our experimentation. As defined earlier in this section, we have used two different lattice instances i.e. Lattice Instance 1 and Lattice Instance 2.

	Sample Size																	
	20%						10%						5%					
	Shlosser		GEE		AE		Shlosser		GEE		AE		Shlosser		GEE		AE	
	Views Const*	Space Const**	Views Const	Space Const	Views Const	Space Const	Views Const	Space Const	Views Const	Space Const	Views Const	Space Const	Views Const	Space Const	Views Const	Space Const	Views Const	Space Const
Lattice Instance 1 (TPCH Database)	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%
	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%
	20	__***	20	__	20	__	20	__	20	__	20	__	20	__	20	__	20	__
Lattice Instance 2 (AANS Database)	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%	5	60%
	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%	10	70%
	20	__	20	__	20	__	20	__	20	__	20	__	20	__	20	__	20	__

* Views Const – Number of views to be materialized constraint
**Space Const – Storage space constraint
***Space Constraint problem instances were limited to two level

Table 3.1: Experimental Parameters Used to Generate Problem Instances

To generate problem instances, estimated rows for each view were obtained by drawing random samples without replacement with three different sample sizes, viz. 20%, 10% and 5%. Here 20% sample size, for example, represents 20% of the total number of rows available in the root view for each lattice instance. As shown in Table 3.1, for each sample size we have employed three different estimators, i.e. the Shlosser Estimator, the GEE and the AE, to estimate the number of rows present in each view in the given lattice instance for both lattice instances.

For each sample size, and within that for each applied estimator, we have solved two versions of the MVS problem i.e. Problems 1 and 2 with equal weights (refer Chapter 2).

For the number of views to be materialized constraint (Problem 1 with equal weights), we have solved it for three different settings, viz. 5 views to be materialized, 10 views to be materialized, and 20 views to be materialized, out of 27 views in a given lattice instance for both lattice instances. The number of views to be materialized includes the root view. So if we are materializing five views, we will find the other four potential views to be materialized besides the root view which is always assumed to be materialized.

For the storage space constraint (Problem 2 with equal weights), we have solved the MVS problems for two different settings. For the first setting, we assumed that 60% of the total space is available for materialization of views, while for the second setting, we assumed that 70% of the total space is available for materialization of views. Here, total space is the space occupied if all the views in a given 27-node lattice instance were materialized.

For example, let 'S' be the total space required if all the views were materialized including the root view, 'r' be the space occupied by the root view, and 'N' be net space available for materializing other views besides the root view. For 60% space constraint, 'N' will be given as:

$N = 0.6*S - r$

Hence, all the other views in a given 27-node lattice instance besides the root view will be considered as potential candidates for selection for materialization till the space 'N' gets filled.

Our extensive empirical evaluation points to the fact that the root view itself occupies almost 40% to 55% of the total space available by itself as it holds the lowest level of aggregated data. This is the main reason we have selected 60% and 70% of the total space as sample size, respectively. As one travels down from the base cuboid to the last cuboid in a given lattice instance (refer to Figure 3.1 and 3.2), the number of rows present in each view decreases by at least one order of magnitude for each level. In some instances, the reduction may be more drastic.

For each lattice instance, we first applied the 0-1 integer programming model to the actual data for each formulations (Problems 1 and 2 with equal weights) to find the cost of optimal solution for various previously mentioned settings (refer to Table 3.1) for each resource constraints. For the same settings, we then applied the HRU heuristics (Problems 1 and 2 with equal weights) to the actual data to find the cost of the solutions for each lattice instance. Then we applied the HRU heuristics to the estimated data for all the previously mentioned settings (refer to Table 3.1) to find the cost of solutions. Finally we compared the cost of solutions obtained by applying the HRU heuristics on the actual as well as on the estimated data with the optimal solution obtained by applying the 0-1 Integer Programming model on the actual data.

So that we can compare the cost of the solutions obtained by the appropriate HRU heuristic with the cost of the optimal solutions obtained by the 0-1 Integer Programming model, we used the following procedure: Let r_i be the actual number of rows present in a view v and $\hat{r}_i$ be the estimated number of rows present in a view v.

1. First, the appropriate HRU heuristic was employed to solve problem instances with r_i. Let us refer to this solution as X_H, which is the set of views to be materialized.
2. Then, the appropriate HRU heuristic was employed to solve problem instances with $\hat{r}_i$. Let us refer to this solution as $\hat{X}_H$, which is the set of views to be materialized.
3. Then, the 0-1 Integer Programming model was employed to solve problem instances with r_i. Let us refer to this solution as X_{IP}, which is the optimal set of views to be materialized.
4. Let $Z(X_H)$, $Z(\hat{X}_H)$ and $Z(X_{IP})$ be the objective function values using r_i and solutions X_H, $\hat{X}_H$ and X_{IP}, respectively.
5. Finally, we compared $Z(X_H)$, $Z(\hat{X}_H)$ and $Z(X_{IP})$ and reported the deviations of cost of solutions for all problem instances (refer to Table 3.1).

In the next section, we report and briefly discuss the experimental results for various mentioned settings.

3.4 Experimental Results

The root nodes of both lattice instances were populated from their respective database in Microsoft's SQL Sever database. Then we applied three estimators (refer to Section 3.2) to estimate the number of rows present in the remainder of the views for both lattice instances. In all estimating procedures, we have used three different sample sizes, viz. 20%, 10% and 5% for drawing random samples without replacement from the respective population (i.e. root view). For each lattice instance, all estimates were computed using the same sample for respective sample sizes. The actual and estimated numbers of rows for both lattice instances for different sample sizes are shown in Tables 3.2 through 3.7. In these tables, we have also shown the Absolute Deviation Proportion (ADP) for these estimates.

From these tables, it appears that all three estimators tend to estimate the number of rows fairly accurately for the views that contain high levels of aggregated data. These methods, in general, tend to lose their accuracy as the number of dimensions and levels in each dimension in the aggregation increases. One can notice the increase in variations in the ADP as the sample size decreases. One reason for this could be as sample size decreases, variation in the respective estimation increases. Also one can notice that in both lattice instances, the variations in the case of the GEE were more while the variation in the case of the AE were the least. One reason for this could be, as mentioned earlier, that the AE adopts itself to the input data distribution. But the GEE performs well only for the data with high skew. In our case, data in both the databases, viz. TPCH and

AANS, are lowly skewed; with TPCH database having large number of low frequency elements while AANS database having small number of low frequency elements.

Node	View	Actual Rows (A)	Shlosser (S)	Abs Dev |A-S|	ADP |A-S|/A	GEE (G)	Abs Dev |A-G|	ADP |A-G|/A	AE (E)	Abs Dev |A-E|	MAD |A-E|/A
0	CPM*	6,001,192	6,001,192	0	0.000	6,001,192	0	0.000	6,001,192	0	0.000
1	CPY	6,001,028	6,001,107	79	0.000	2,683,791	3,317,237	0.553	6,001,192	164	0.000
2	NPM	5,710,004	5,858,214	148,210	0.026	2,642,931	3,067,073	0.537	6,001,192	291,188	0.051
3	CTM	5,873,146	5,937,209	64,063	0.011	2,665,567	3,207,579	0.546	6,001,192	128,046	0.022
4	NPY	5,510,833	5,753,234	242,401	0.044	2,612,666	2,898,167	0.526	6,001,192	490,359	0.089
5	CTY	5,746,207	5,873,933	127,726	0.022	2,647,446	3,098,761	0.539	6,001,192	254,985	0.044
6	NTM	45,000	45,000	0	0.000	45,000	0	0.000	45,000	0	0.000
7	NTY	26,250	26,250	0	0.000	26,250	0	0.000	26,250	0	0.000
8	CP	6,000,127	6,000,570	443	0.000	2,683,638	3,316,489	0.553	6,001,192	1,065	0.000
9	PM	2,199,959	3,337,699	1,137,740	0.517	1,841,899	358,060	0.163	2,723,921	523,962	0.238
10	CM	1,126,591	1,680,661	554,070	0.492	1,164,430	37,839	0.034	1,224,216	97,625	0.087
11	NP	3,494,158	4,510,793	1,016,635	0.291	2,236,494	1,257,664	0.360	5,473,028	1,978,870	0.566
12	PY	1,378,602	2,266,261	887,659	0.644	1,429,477	50,875	0.037	1,616,643	238,041	0.173
13	CY	617,446	793,585	176,139	0.285	675,558	58,112	0.094	642,939	25,493	0.041
14	NM	300	300	0	0.000	300	0	0.000	300	0	0.000
15	CT	4,804,917	5,355,939	551,022	0.115	2,496,101	2,308,816	0.481	6,001,192	1,196,275	0.249
16	TM	1,800	1,800	0	0.000	1,800	0	0.000	1,800	0	0.000
17	NY	175	175	0	0.000	175	0	0.000	175	0	0.000
18	NT	3,750	3,750	0	0.000	3,750	0	0.000	3,750	0	0.000
19	TY	1,050	1,050	0	0.000	1,050	0	0.000	1,050	0	0.000
20	C	99,996	100,250	254	0.003	100,723	727	0.007	100,136	140	0.001
21	P	199,996	201,990	1,994	0.010	203,213	3,217	0.016	200,754	758	0.004
22	M	12	12	0	0.000	12	0	0.000	12	0	0.000
23	N	25	25	0	0.000	25	0	0.000	25	0	0.000
24	T	150	150	0	0.000	150	0	0.000	150	0	0.000
25	Y	7	7	0	0.000	7	0	0.000	7	0	0.000
26	ALL*	1	1	0	0.000	1	0	0.000	1	0	0.000
	Sum				2.459			4.445			1.566

* Values are already known

Table 3.2: Actual and Estimated Number of Rows for Lattice Instance 1 for Sample Size of 20%

Node	View	Actual Rows (A)	Shlosser (S)	Abs Dev |A-S|	ADP |A-S|/A	GEE (G)	Abs Dev |A-G|	ADP |A-G|/A	AE (E)	Abs Dev |A-E|	ADP |A-E|/A
0	CPM*	6,001,192	6,001,192	0	0.000	6,001,192	0	0.000	6,001,192	0	0.000
1	CPY	6,001,028	6,001,192	164	0.000	1,897,745	4,103,283	0.684	6,001,192	164	0.000
2	NPM	5,710,004	5,920,750	210,746	0.037	1,881,898	3,828,106	0.670	6,001,192	291,188	0.051
3	CTM	5,873,146	5,966,203	93,057	0.016	1,890,862	3,982,284	0.678	6,001,192	128,046	0.022
4	NPY	5,510,833	5,862,811	351,978	0.064	1,870,433	3,640,400	0.661	6,001,192	490,359	0.089
5	CTY	5,746,207	5,931,072	184,865	0.032	1,883,936	3,862,271	0.672	6,001,192	254,985	0.044
6	NTM	45,000	45,002	2	0.000	45,004	4	0.000	45,000	0	0.000
7	NTY	26,250	26,250	0	0.000	26,250	0	0.000	26,250	0	0.000
8	CP	6,000,127	6,000,894	767	0.000	1,897,686	4,102,441	0.684	6,001,192	1,065	0.000
9	PM	2,199,959	4,295,787	2,095,828	0.953	1,541,146	658,813	0.299	2,617,761	417,802	0.190
10	CM	1,126,591	2,692,036	1,565,445	1.390	1,150,181	23,590	0.021	1,154,898	28,307	0.025
11	NP	3,494,158	5,118,531	1,624,373	0.465	1,718,979	1,775,179	0.508	5,339,332	1,845,174	0.528
12	PY	1,378,602	3,392,439	2,013,837	1.461	1,329,734	48,868	0.035	1,569,089	190,487	0.138
13	CY	617,446	1,413,377	795,931	1.289	763,687	146,241	0.237	623,693	6,247	0.010
14	NM	300	300	0	0.000	300	0	0.000	300	0	0.000
15	CT	4,804,917	5,633,370	828,453	0.172	1,824,588	2,980,329	0.620	6,001,192	1,196,275	0.249
16	TM	1,800	1,800	0	0.000	1,800	0	0.000	1,800	0	0.000
17	NY	175	175	0	0.000	175	0	0.000	175	0	0.000
18	NT	3,750	3,750	0	0.000	3,750	0	0.000	3,750	0	0.000
19	TY	1,050	1,050	0	0.000	1,050	0	0.000	1,050	0	0.000
20	C	99,996	107,730	7,734	0.077	109,729	9,733	0.097	101,304	1308	0.013
21	P	199,996	282,162	82,166	0.411	254,149	54,153	0.271	210,225	102,29	0.051
22	M	12	12	0	0.000	12	0	0.000	12	0	0.000
23	N	25	25	0	0.000	25	0	0.000	25	0	0.000
24	T	150	150	0	0.000	150	0	0.000	150	0	0.000
25	Y	7	7	0	0.000	7	0	0.000	7	0	0.000
26	ALL*	1	1	0	0.000	1	0	0.000	1	0	0.000
	Sum				6.367			6.138			1.411

* Values are already known

Table 3.3: Actual and Estimated Number of Rows for Lattice Instance 1 for Sample Size of 10%

Node	View	Actual Rows (A)	Shlosser (S)	Abs Dev |A-S|	ADP |A-S|/A	GEE (G)	Abs Dev |A-G|	ADP |A-G|/A	AE (E)	Abs Dev |A-E|	ADP |A-E|/A
0	CPM*	6,001,192	6,001,192	0	0.000	6,001,192	0	0.000	6,001,192	0	0.000
1	CPY	6,001,028	6,001,192	164	0.000	1,341,908	4,659,120	0.776	6,001,192	164	0.000
2	NPM	5,710,004	5,958,474	248,470	0.044	1,335,952	4,374,052	0.766	6,001,192	291,188	0.051
3	CTM	5,873,146	5,982,663	109,517	0.019	1,339,326	4,533,820	0.772	6,001,192	128,046	0.022
4	NPY	5,510,833	5,930,291	419,458	0.076	1,332,012	4,178,821	0.758	6,001,192	490,359	0.089
5	CTY	5,746,207	5,964,613	218,406	0.038	1,336,808	4,409,399	0.767	6,001,192	254,985	0.044
6	NTM	45,000	46,122	1,122	0.025	46,311	1,311	0.029	45,078	78	0.002
7	NTY	26,250	26,259	9	0.000	26,267	17	0.001	26,252	2	0.000
8	CP	6,000,127	6,001,078	951	0.000	1,341,892	4,658,235	0.776	6,001,192	1,065	0.000
9	PM	2,199,959	5,022,564	2,822,605	1.283	1,201,010	998,949	0.454	2,529,270	329,311	0.150
10	CM	1,126,591	3,856,728	2,730,137	2.423	1,018,658	107,933	0.096	1,077,018	49,573	0.044
11	NP	3,494,158	5,523,824	2,029,666	0.581	1,274,377	2,219,781	0.635	5,294,604	1,800,446	0.515
12	PY	1,378,602	4,418,421	3,039,819	2.205	1,108,624	269,978	0.196	1,496,721	118,119	0.086
13	CY	617,446	2,556,057	1,938,611	3.140	788,230	170,784	0.277	582,933	34,513	0.056
14	NM	300	300	0	0.000	300	0	0.000	300	0	0.000
15	CT	4,804,917	5,803,177	998,260	0.208	1,314,156	3,490,761	0.726	6,001,192	1,196,275	0.249
16	TM	1,800	1,800	0	0.000	1,800	0	0.000	1,800	0	0.000
17	NY	175	175	0	0.000	175	0	0.000	175	0	0.000
18	NT	3,750	3,750	0	0.000	3,750	0	0.000	3,750	0	0.000
19	TY	1,050	1,050	0	0.000	1,050	0	0.000	1,050	0	0.000
20	C	99,996	198,386	98,390	0.984	152,001	52,005	0.520	104,579	4,583	0.046
21	P	199,996	833,421	633,425	3.167	387,862	187,866	0.939	227,855	27,859	0.139
22	M	12	12	0	0.000	12	0	0.000	12	0	0.000
23	N	25	25	0	0.000	25	0	0.000	25	0	0.000
24	T	150	150	0	0.000	150	0	0.000	150	0	0.000
25	Y	7	7	0	0.000	7	0	0.000	7	0	0.000
26	ALL*	1	1	0	0.000	1	0	0.000	1	0	0.000
	Sum				14.19			8.490			1.493

* Values are already known

Table 3.4: Actual and Estimated Number of Rows for Lattice Instance 1 for Sample Size of 5%

Node	Views	Actual Rows (A)	Shlosser (S)	Abs Dev |A-S|	ADP |A-S|/A	GEE (G)	Abs Dev |A-G|	ADP |A-G|/A	AE (E)	Abs Dev |A-E|	ADP |A-E|/A
0	CPM*	9,275,377	9,275,377	0	0	9,275,377	0	0	9,275,377	0	0
1	CPY	3,722,050	5,050,036	1,327,986	0.357	2,801,702	920,348	0.247	4,289,960	567,910	0.153
2	NPM	1,208,341	1,246,332	242,887	0.242	1,047,691	44,246	0.044	1,016,862	13,417	0.013
3	CTM	406,941	1,570,251	361,910	0.3	1,246,307	37,966	0.031	1,238,163	29,822	0.025
4	NPY	1,003,445	352,443	25,472	0.078	344,281	17,310	0.053	329,426	2,455	0.008
5	CTY	326,971	451,995	45,054	0.111	438,316	31,375	0.077	416,636	9,695	0.024
6	NTM	73,697	74,295	598	0.008	75,125	1,428	0.019	73,939	242	0.003
7	NTY	61,055	61,436	381	0.006	62,052	997	0.016	61,178	123	0.002
8	CP	609,926	729,032	119,106	0.195	676,665	66,739	0.109	639,527	29,601	0.049
9	PM	21,576	17,980	0	0	17,980	0	0	17,980	0	0
10	CM	53,002	53,005	3	0	53,066	64	0.001	53,002	0	0
11	NP	110,228	1,200	0	0	1,200	0	0	1,200	0	0
12	PY	17,980	43,549	54	0.001	43,841	346	0.008	43,535	40	0.001
13	CY	43,495	111,299	1,071	0.01	112,822	2,594	0.024	110,749	521	0.005
14	NM	840	841	1	0.001	841	1	0.001	863	23	0.027
15	CT	34,223	21,576	0	0	21,576	0	0	21,576	0	0
16	TM	1,200	6,151	1	0	6,152	2	0	6,153	3	0
17	NY	699	34,238	15	0	34,264	41	0.001	34,237	14	0
18	NT	6,150	699	0	0	699	0	0	699	0	0
19	TY	1,000	1,000	0	0	1,000	0	0	1,000	0	0
20	C	4,417	4,417	0	0	4,417	0	0	4,417	0	0
21	P	1,798	12	0	0	12	0	0	12	0	0
22	M	12	100	0	0	100	0	0	100	0	0
23	N	70	1,798	0	0	1,798	0	0	1,798	0	0
24	T	100	70	0	0	70	0	0	70	0	0
25	Y	10	10	0	0	10	0	0	10	0	0
26	ALL*	1	1	0	0	1	0	0	1	0	0
	Sum				1.309			0.633			0.31

* Values are already known

Table 3.5: Actual and Estimated Number of Rows for Lattice Instance 2 for Sample Size of 20%

Node	Views	Actual Rows (A)	Shlosser (S)	Abs Dev |A-S|	ADP |A-S|/A	GEE (G)	Abs Dev |A-G|	ADP |A-G|/A	AE (E)	Abs Dev |A-E|	ADP |A-E|/A
0	CPM*	9,275,377	9,275,377	0	0	9,275,377	0	0	9,275,377	0	0
1	CPY	3,722,050	6,450,148	2,728,098	0.733	2,339,522	1,382,528	0.371	3,879,718	157,668	0.042
2	NPM	1,208,341	2,110,140	1,106,695	1.103	1,153,541	150,096	0.15	953,626	49,819	0.05
3	CTM	406,941	2,619,232	1,410,891	1.168	1,325,427	117,086	0.097	1,147,385	60,956	0.05
4	NPY	1,003,445	462,005	135,034	0.413	393,809	66,838	0.204	319,085	7,886	0.024
5	CTY	326,971	644,579	237,638	0.584	508,797	101,856	0.25	405,828	1,113	0.003
6	NTM	73,697	76,950	3,253	0.044	80,237	6,540	0.089	73,488	209	0.003
7	NTY	61,055	63,077	2,022	0.033	65,938	4,883	0.08	60,969	86	0.001
8	CP	609,926	1,186,693	576,767	0.946	787,416	177,490	0.291	623,295	13,369	0.022
9	PM	21,576	17,980	0	0	17,980	0	0	17,980	0	0
10	CM	53,002	53,408	406	0.008	54,103	1,101	0.021	53,096	94	0.002
11	NP	110,228	1,200	0	0	1,200	0	0	1,200	0	0
12	PY	17,980	43,889	394	0.009	45,196	1,701	0.039	43,397	98	0.002
13	CY	43,495	117,359	7,131	0.065	122,197	11,969	0.109	110,176	52	0
14	NM	840	839	1	0.001	839	1	0.001	839	1	0.001
15	CT	34,223	21,576	0	0	21,576	0	0	21,576	0	0
16	TM	1,200	6,152	2	0	6,163	13	0.002	6,150	0	0
17	NY	699	34,383	160	0.005	34,693	470	0.014	34,275	52	0.002
18	NT	6,150	699	0	0	699	0	0	699	0	0
19	TY	1,000	1,000	0	0	1,000	0	0	1,000	0	0
20	C	4,417	4,417	0	0	4,417	0	0	4,417	0	0
21	P	1,798	12	0	0	12	0	0	12	0	0
22	M	12	100	0	0	100	0	0	100	0	0
23	N	70	1,798	0	0	1,798	0	0	1,798	0	0
24	T	100	70	0	0	70	0	0	70	0	0
25	Y	10	10	0	0	10	0	0	10	0	0
26	ALL*	1	1	0	0	1	0	0	1	0	0
	Sum				5.111			1.718			0.203

* Values are already known

Table 3.6: Actual and Estimated Number of Rows for Lattice Instance 2 for Sample Size of 10%

Node	Views	Actual Rows (A)	Shlosser (S)	Abs Dev |A-S|	ADP |A-S|/A	GEE (G)	Abs Dev |A-G|	ADP |A-G|/A	AE (E)	Abs Dev |A-E|	ADP |A-E|/A
0	CPM*	9,275,377	9,275,377	0	0	9,275,377	0	0	9,275,377	0	0
1	CPY	3,722,050	7,616,304	3,894,254	1.046	1,834,523	1,887,527	0.507	3,600,477	121,573	0.033
2	NPM	1,208,341	3,781,522	2,778,077	2.769	1,186,226	182,781	0.182	870,772	132,673	0.132
3	CTM	406,941	4,366,060	3,157,719	2.613	1,297,470	89,129	0.074	1,033,276	175,065	0.145
4	NPY	1,003,445	858,533	531,562	1.626	467,840	140,869	0.431	294,015	32,956	0.101
5	CTY	326,971	1,286,073	879,132	2.16	604,712	197,771	0.486	376,842	30,099	0.074
6	NTM	73,697	95,670	21,973	0.298	95,681	21,984	0.298	72,556	1,141	0.015
7	NTY	61,055	73,847	12,792	0.21	76,558	15,503	0.254	59,959	1,096	0.018
8	CP	609,926	2,382,030	1,772,104	2.905	889,752	279,826	0.459	576,479	33,447	0.055
9	PM	21,576	17,980	0	0	17,980	0	0	17,980	0	0
10	CM	53,002	59,295	6,293	0.119	61,362	8,360	0.158	53,478	476	0.009
11	NP	110,228	1,200	0	0	1,200	0	0	1,200	0	0
12	PY	17,980	48,159	4,664	0.107	50,848	7,353	0.169	43,244	251	0.006
13	CY	43,495	165,457	55,229	0.501	150,021	39,793	0.361	108,596	1,632	0.015
14	NM	840	838	2	0.002	838	2	0.002	838	2	0.002
15	CT	34,223	21,576	0	0	21,576	0	0	21,576	0	0
16	TM	1,200	6,155	5	0.001	6,219	69	0.011	6,139	11	0.002
17	NY	699	36,369	2,146	0.063	37,965	3,742	0.109	34,473	250	0.007
18	NT	6,150	702	3	0.004	709	10	0.014	704	5	0.007
19	TY	1,000	1,000	0	0	1,000	0	0	1,000	0	0
20	C	4,417	4,417	0	0	4,417	0	0	4,417	0	0
21	P	1,798	12	0	0	12	0	0	12	0	0
22	M	12	100	0	0	100	0	0	100	0	0
23	N	70	1,798	0	0	1,798	0	0	1,798	0	0
24	T	100	70	0	0	70	0	0	70	0	0
25	Y	10	10	0	0	10	0	0	10	0	0
26	ALL*	1	1	0	0	1	0	0	1	0	0
	Sum				14.424			3.516			0.621

* Values are already known

Table 3.7: Actual and Estimated Number of Rows for Lattice Instance 2 for Sample Size of 5%

Also one can notice that for all sample sizes, variations were less in the case of AANS database compared to TPCH database. One reason for these could be that AANS database data is comparatively less skewed than the data in TPCH database or it could be the presence of small number of low frequency elements in AANS database. Also, as discussed earlier, TPCH database for 1 GB size does not represent a realistic picture. For example, in seven years of data, no customer has ordered the same product more than once. Because of this, some of the derived views have the same number of rows as their immediate parent.

We have tabulated the cost of optimal and heuristic solutions for both lattice instances for different sample sizes viz. 20%, 10% and 5%, and resource constraints. Tables 3.8 through 3.10 tabulate the results for the number of views to be materialized constraint. We have solved the MVS problem (Problem 1 with equal weights) for three different problem instances. In the first problem instance, we imposed a constraint of 5 views to be materialized; in the second problem instance, we set it to 10 views; in the third problem instance, we set it to 20 views.

As one can observe from these tables (Table 3.8, 3.9 and 3.10), the heuristic failed to identify the optimal solutions in certain cases. That means that in these cases it recommended a materialization scheme that is different from the optimal materialization scheme. One interesting point to notice here is that even a sample size as low as 5% produced acceptable results and hence it was not necessary to run 10% or 20% sample size experiments especially if we use AE method for sampling.

Lattice Instance	Problem Instance	Optimal Solution (Set of Materialized Views)	Solutions by HRU Heuristic 1: Set of Materialized Views			
			With Actual Number of Rows	With Shlosser Estimates	With GEE Estimates	With AE Estimates
1: TPCH	1: Materialize 5 views	0,6,9,10,11	0,6,9,10,11	0,6,9,10,21	0,1,2,6,10	0,6,9,10,21
	2: Materialize 10 views	0,2,6,9,10,11, 12,15,20,21	0,2,6,9,10,11, 12,15,20,21	0,6,9,10,11,12, 13,15,20,21	0,1,2,3,6,9, 10,13,20,21	0,6,9,10,11,12, 13,14,20,21
	3: Materialize 20 views	0,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16,18,20,21,24	0,2,3,4,5,6,7,8,9, 10,11,12,13,14,15 ,16,18,20,21,24	0,2,3,4,5,6,7,9,10, 11,12,13,14,15,16 ,18,19,20,21,24	0,1,2,3,4,5,6,7,9, 10,11,12,13,14,15 ,16,18,20,21,24	0,6,7,9,10,11,12,13, 14,16,17,18,19,20,21 ,22,23,24,25,26
2: AANS	1: Materialize 5 views	0,1,2,3,9	0,1,2,3,9	0,2,3,8,9	0,1,2,3,9	0,1,2,3,9
	2: Materialize 10 views	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15
	3: Materialize 20 views	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21	0,1,2,3,4,5,6,7,8,9 ,10,11,12,13,14,1 5,16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21

Table 3.8: Views to be Materialized for Three Problems in Lattice Instances 1 and 2 for Sample Size of 20% Under the Number of Views to be Materialized Constraint

Lattice Instance	Problem Instance	Optimal Solution (Set of Materialized Views)	Solutions by HRU Heuristic 1: Set of Materialized Views			
			With Actual Number of Rows	With Shlosser Estimates	With GEE Estimates	With AE Estimates
1: TPCH	1: Materialize 5 views	0,6,9,10,11	0,6,9,10,11	0,6,9,10,21	0,1,2,3,6	0,6,9,10,21
	2: Materialize 10 views	0,2,6,9,10,11, 12,15,20,21	0,2,6,9,10,11, 12,15,20,21	0,6,9,10,11,12, 13,15,20,21	0,1,2,3,6,9, 10,13,20,21	0,6,9,10,11,12, 13,14,20,21
	3: Materialize 20 views	0,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16,18,20,21,24	0,2,3,4,5,6,7,8,9, 10,11,12,13,14,15 ,16,18,20,21,24	0,2,3,4,5,6,7,9,10, 11,12,13,14,15,16 ,18,19,20,21,24	0,1,2,3,4,5,6,7,9, 10,11,12,13,14,15 ,16,18,20,21,24	0,6,7,9,10,11,12,13, 14,16,17,18,19,20,21 ,22,23,24,25,26
2: AANS	1: Materialize 5 views	0,1,2,3,9	0,1,2,3,9	0,2,3,8,9	0,1,2,3,9	0,1,2,3,9
	2: Materialize 10 views	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15	0,1,2,3,6,8, 9,10,11,15
	3: Materialize 20 views	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14,15, 16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21	0,1,2,3,4,5,6,7,8,9 ,10,11,12,13,14,1 5,16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21

Table 3.9: Views to be Materialized for Three Problems in Lattice Instances 1 and 2 for Sample Size of 10% Under the Number of Views to be Materialized Constraint

Lattice Instance	Problem Instance	Optimal Solution (Set of Materialized Views)	Solutions by HRU Heuristic 1: Set of Materialized Views			
			With Actual Number of Rows	With Shlosser Estimates	With GEE Estimates	With AE Estimates
1: TPCH	1: Materialize 5 views	0,6,9,10,11	0,6,9,10,11	0,6,13,20,21	0,1,2,3,6	0,6,9,10,21
	2: Materialize 10 views	0,2,6,9,10, 11,12,15,20,21	0,2,6,9,10, 11,12,15,20,21	0,6,9,10,11, 12,13,14,20,21	0,1,2,3,6, 9,10,14,20,21	0,6,9,10,11, 12,13,14,20,21
	3: Materialize 20 views	0,2,3,4,5,6,7,8,9, 10,11,12,13,14,15, 16,18,20,21,24	0,2,3,4,5,6,7,8,9, 10,11,12,13,14,15 ,16,18,20,21,24	0,2,3,4,5,6,7,9,10, 11,12,13,14,15,16 ,18,19,20,21,24	0,1,2,3,4,5,6,7,9, 10,11,12,13,14,15 ,16,18,20,21,24	0,6,7,9,10,11,12,13, 14,16,17,18,19,20,21 ,22,23,24,25,26
2: AANS	1: Materialize 5 views	0,1,2,3,9	0,1,2,3,9	0,2,3,8,9	0,1,2,3,9	0,1,2,3,9
	2: Materialize 10 views	0,1,2,3,6, 8,9,10,11,15	0,1,2,3,6, 8,9,10,11,15	0,1,2,3,6, 8,9,10,11,15	0,1,2,3,6, 8,9,10,11,15	0,1,2,3,6, 8,9,10,11,15
	3: Materialize 20 views	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14,15, 16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21	0,1,2,3,4,5,6,7,8,9 ,10,11,12,13,14,1 5,16,18,20,21	0,1,2,3,4,5,6,7,8, 9,10,11,12,13,14, 15,16,18,20,21

Table 3.10: Views to be Materialized for Three Problems in Lattice Instances 1 and 2 for Sample Size of 5% Under the Number of Views to be Materialized Constraint

Tables 3.11 through 3.13 tabulate the results for both lattice instances for different sample sizes, viz. 20%, 10% and 5%, for the storage space constraint. We have solved the MVS problem (Problem 2 with equal weights) for two different problem instances: in first problem instance, we assume that only 60% of the total space is available for materialization of views, while in second problem instance, we assumed this percentage to be 70%.

Lattice Instance	Problem Instance	Optimal Solution (Set of Materialized Views)	Solutions by HRU Heuristic 2: Set of Materialized Views			
			With Actual Number of Rows	With Shlosser Estimates	With GEE Estimates	With AE Estimates
1: TPCH	1: 60% of Total Space	0,2,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,5,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,1,2,3,6,7,9, 10,13,14,16,17, 18,19,20,21,22, 23,24,25,26	0,6,7,9,10,11, 12, 13,14,16,17, 18,19,20,21,22, 23, 24,25,26
	2: 70% of Total Space	0,2,3,4,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,4,5,6,7,9,10, 11,12,13,14,15,16 ,17,18,19,20,21,2 2,23,24,25,26	0,2,4,5,6,7,9,10, 11,12,13,14,15,16 ,17,18,19,20,21, 22,23,24,25,26	0,1,2,3,6,7,9,10, 11,12,13,14,16,17 ,18,19,20,21,22, 23,24,25,26	0,6,7,9,10,11, 12,13,14,16,17, 18,19,20,21,22, 23,24,25,26
2: AANS	1: 60% of Total Space	0,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,3,6,9,10,11, 14,15,16,18, 20,21,22,23, 24,25,26	0,6,7,9,10,11,12, 13,14,15,16,17, 18,19,20,21,22, 23,24,25,26	0,3,6,9,10,11, 14,15,16,18, 20,21,22,23, 24,25,26
	2: 70% of Total Space	0,2,3,6,7,8,9,10, 11,12,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,3,6,8,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,3,6,8,9,10, 11,14,15,16, 18, 20,21,22, 23,24,25,26	0,2,3,6,7,9,10, 11,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,3,5,6,7,9, 10,11,12,15,16, 17,19,21,22,23, 24,25,26

Table 3.11: Views to be Materialized for Two Problems in Lattice Instances 1 and 2 for Sample Size of 20% Under the Storage Space Constraint

Lattice Instance	Problem Instance	Optimal Solution (Set of Materialized Views)	Solutions by HRU Heuristic 2: Set of Materialized Views			
			With Actual Number of Rows	With Shlosser Estimates	With GEE Estimates	With AE Estimates
1: TPCH	1: 60% of Total Space	0,2,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,5,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21,22 ,23,24,25,26	0,2,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,1,2,3,6,7,9,10, 13,14,16,17, 18,19,20,21,22, 23,24,25,26	0,6,7,9,10,11, 12, 13,14,16,17, 18,19,20,21,22, 23, 24,25,26
	2: 70% of Total Space	0,2,3,4,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,4,5,6,7,9,10, 11,12,13,14,15,16 ,17,18,19,20,21, 22,23,24,25,26	0,2,5,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,1,2,3,6,7,9,10, 12,13,14,16,17, 18,19,20,21,22, 23,24,25,26	0,6,7,9,10,11,12, 13,14,16,17,18, 19,20,21,22,23, 24,25,26
2: AANS	1: 60% of Total Space	0,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,3,6,9,10,11, 14,15,16,18, 20,21,22,23, 24,25,26	0,6,7,9,10,11,12,1 3,14,15,16,17, 18,19,20,21,22, 23,24,25,26	0,3,6,7,9,10,11, 12,13,14,15,16,17, 18,19,20,21,22,23, 24,25,26
	2: 70% of Total Space	0,2,3,6,7,8,9,10, 11,12,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,3,6,8,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,3,6,8,9,10, 11,14,15,16, 18, 20,21,22, 23,24,25,26	0,3,5,6,7,8,9,10, 11,14,15,16,17 ,18,19,20,21,22, 23,24,25,26	0,2,3,6,8,9,10, 11,13,14,15, 16,18,20,21,22, 23,24,25,26

Table 3.12: Views to be Materialized for Two Problems in Lattice Instances 1 and 2 for Sample Size of 10% Under the Storage Space Constraint

Lattice Instance	Problem Instance	Optimal Solution (Set of Materialized Views)	Solutions by HRU Heuristic 2: Set of Materialized Views			
			With Actual Number of Rows	With Shlosser Estimates	With GEE Estimates	With AE Estimates
1: TPCH	1: 60% of Total Space	0,2,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,5,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,5,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,1,2,3,6,7,10, 13,14,16,17, 18,19,20,21,22, 23,24,25,26	0,6,7,9,10,11, 12, 13,14,16,17, 18,19,20,21,22, 23, 24,25,26
	2: 70% of Total Space	0,2,3,4,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,4,5,6,7,9,10, 11,12,13,14,15,16, 17,18,19,20,21,22, 23,24,25,26	0,4,5,6,7,9,10,11, 12,13,14,15,16,1 7,18,19,20,21,22, 23,24,25, 26	0,1,2,3,6,7,9,10, 12,13,14,16,17, 18,19,20,21,22, 23,24,25,26	0,6,7,9,10,11, 12,13,14,16,17, 18,19,20,21,22, 23,24,25,26
2: AANS	1: 60% of Total Space	0,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,3,6,7,9,10,11, 12,13,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,3,6,9,10,11, 14,15,16,18, 20,21,22,23, 24,25,26	0,6,7,9,10,13, 14,15,16,17,18, 19,20,21,22, 23,24,25,26	0,6,7,9,10,11,12, 13,14,15,16,17, 18,19,20,21,22, 23,24,25,26
	2: 70% of Total Space	0,2,3,6,7,8,9,10, 11,12,14,15,16, 17,18,19,20,21, 22,23,24,25,26	0,2,3,6,8,9,10,11, 12,13,14,15,16, 17,18,19,20,21,22, 23,24,25,26	0,2,3,6,8,9,10, 11,14,15,16, 18, 20,21,22, 23,24,25,26	0,3,6,7,8,9,10,11, 12,14,15,16,17, 18,19,20,21,22, 23,24,25,26	0,2,3,5,6,7,9,10, 11,12,13,14,15, 16,17,18,19,20,21, 22,23,24,25,26

Table 3.13: Views to be Materialized for Two Problems in Lattice Instances 1 and 2 for Sample Size of 5% Under the Storage Space Constraint

As one can see from these tables (Tables 3.11 through 3.13), in some cases heuristic gave the optimal solutions while in other cases it failed to produce the optimal solutions and recommended a different materialization scheme than the optimal view materialization scheme. Again one can notice the same trend as observed in the case of the number of views to be materialized constraint. Even the sample size as low as 5% produced acceptable results making sample of sizes 10% and 20% somewhat unnecessary except for the most important applications.

Appropriate comparisons between the two approaches call for calculating the following objective function values and making a critical evaluation of the differences. Let Z($\hat{X}_{Shl1}$) be the objective function value using actual data but using solution obtained by solving problem instances generated using the Schlosser estimates for Problem 1 with equal weights using HRU Heuristic 1. Similarly, Z($\hat{X}_{GEE1}$) and Z($\hat{X}_{AE1}$) are defined. Let

$Z(X_{IP1})$ be the objective function value using actual data obtained by solving problem instances generated using the 0-1 Integer Programming model and $Z(X_{HRU1})$ be the objective function value using actual data but using solution obtained by HRU Heuristic 1 with actual data for Problem 1 with equal weights. Tables 3.14 through 3.16 present the values for $Z(X_{HRU1})$, $Z(\hat{X}_{Shl1})$, $Z(\hat{X}_{GEE1})$, $Z(\hat{X}_{AE1})$ along with corresponding deviations from $Z(X_{IP1})$ for three problem instances for both lattice instances for the number of views to be materialized constraint .

One way to look at these deviations is to split it into two components, the difference caused by the use of estimated data and by the use of heuristics as defined in the following equation, where the first part in the RHS of the equation is the difference caused by the use of estimated data and the second part is caused by the use of heuristics.

$$Z(\hat{X}_{Shl1}) - Z(X_{IP1}) = (Z(\hat{X}_{Shl1}) - Z(X_{HRU1})) + (Z(X_{HRU1}) - Z(X_{IP1}))$$

			Cost of Solution with Real Data		**Cost of Solution With Estimated Data Using HRU Heuristic 1**					
Lattice Instance	**Problem Instance**	**Optimal Cost** $Z(X_{IP1})$	**HRU Heuristic 1** $Z(X_{HRU1})$	**% Dev***	**Shlosser** $Z(\hat{X}_{Shl1})$	**% Dev***	**GEE** $Z(\hat{X}_{GEE1})$	**% Dev***	**AE** $Z(\hat{X}_{AE1})$	**% Dev***
1	1	62,023,340	62,023,340	0.00	62,530,420	0.82	74,186,290	19.61	62,530,420	0.82
	2	56,396,780	56,396,780	0.00	56,470,010	0.13	59,736,640	5.92	57,398,080	1.78
	3	54,844,770	54,844,766	0.00	54,845,080	0.00	54,845,500	0.00	57,204,800	4.30
2	1	25,421,980	25,421,980	0.00	27,264,760	7.25	25,421,980	0.00	25,421,980	0.00
	2	17,638,620	17,638,620	0.00	17,638,620	0.00	17,638,620	0.00	17,638,620	0.00
	3	16,989,310	16,989,312	0.00	16,989,310	0.00	16,989,310	0.00	16,989,310	0.00

**** % Dev*** stands for the deviation of the cost of solution (in % of the optimal cost)

Table 3.14: Deviation of the Cost of Heuristic Solution from the Optimal Solution for Sample Size of 20% Under the Number of Views to be Materialized Constraint

Lattice Instance	Problem Instance	Optimal Cost $Z(X_{IP1})$	Cost of Solution with Real Data		Cost of Solution With Estimated Data Using HRU Heuristic 1					
			HRU Heuristic 1 $Z(X_{HRU1})$	% Dev*	Shlosser $Z(\hat{X}_{Shl1})$	% Dev*	GEE $Z(\hat{X}_{GEE1})$	% Dev*	AE $Z(\hat{X}_{AE1})$	% Dev*
1	1	62,023,340	62,023,340	0.00	62,530,420	0.82	88,042,150	41.95	62,530,420	0.82
	2	56,396,780	56,396,780	0.00	56,470,010	0.13	59,736,640	5.92	57,398,080	1.78
	3	54,844,770	54,844,766	0.00	54,845,080	0.00	54,845,500	0.00	57,204,800	4.30
2	1	25,421,980	25,421,980	0.00	27,264,760	7.25	25,421,980	0.00	25,421,980	0.00
	2	17,638,620	17,638,620	0.00	17,638,620	0.00	17,638,620	0.00	17,638,620	0.00
	3	16,989,310	16,989,312	0.00	16,989,310	0.00	16,989,310	0.00	16,989,310	0.00

% Dev stands for the deviation of the cost of solution (in % of the optimal cost)

Table 3.15: Deviation of the Cost of Heuristic Solution from the Optimal Solution for Sample Size of 10% Under the Number of Views to be Materialized Constraint

Lattice Instance	Problem Instance	Optimal Cost $Z(X_{IP1})$	Cost of Solution with Real Data		Cost of Solution With Estimated Data Using HRU Heuristic 1					
			HRU Heuristic 1 $Z(X_{HRU1})$	% Dev*	Shlosser $Z(\hat{X}_{Shl1})$	% Dev*	GEE $Z(\hat{X}_{GEE1})$	% Dev*	AE $Z(\hat{X}_{AE1})$	% Dev*
1	1	62,023,340	62,023,340	0.00	73,471,740	18.46	88,042,150	41.95	62,530,420	0.82
	2	56,396,780	56,396,780	0.00	57,398,080	1.78	59,977,590	6.35	57,398,080	1.78
	3	54,844,770	54,844,766	0.00	54,845,080	0.00	54,845,500	0.00	57,204,800	4.30
2	1	25,421,980	25,421,980	0.00	27,264,760	7.25	25,421,980	0.00	25,421,980	0.00
	2	17,638,620	17,638,620	0.00	17,638,620	0.00	17,638,620	0.00	17,638,620	0.00
	3	16,989,310	16,989,312	0.00	16,989,310	0.00	16,989,310	0.00	16,989,310	0.00

% Dev stands for the deviation of the cost of solution (in % of the optimal cost)

Table 3.16: Deviation of the Cost of Heuristic Solution from the Optimal Solution for Sample Size of 5% Under the Number of Views to be Materialized Constraint

Let Z($\hat{X}_{Shl2}$) be the objective function value using actual data but using solution obtained by solving problem instances generated using the Schlosser estimates for Problem 2 with equal weights using HRU Heuristic 2. Similarly, Z($\hat{X}_{GEE2}$) and Z($\hat{X}_{AE2}$) are defined. Let *Z(X_{IP2})* be the objective function value using actual data obtained by solving problem instances generated using the 0-1 Integer Programming model and Z(X_{HRU2}) be the objective function value using actual data but using solution obtained by HRU Heuristic 2 with actual data for Problem 2 with equal weights. Tables 3.17 through 3.19 present the values for Z(X_{HRU2}), Z($\hat{X}_{Shl2}$), Z($\hat{X}_{GEE2}$), Z($\hat{X}_{AE2}$) along with corresponding deviations from *Z(X_{IP2}))* for two problem instances for both lattice instances for the storage space constraint .

			Cost of Solution with Real Data		**Cost of Solution With Estimated Data Using HRU Heuristic 2**					
Lattice Instance	**Problem Instance**	**Optimal Cost** Z(X_{IP2})	**HRU Heuristic 2** Z(X_{HRU2})	**% Dev***	**Shlosser** Z($\hat{X}_{Shl2}$)	**% Dev***	**GEE** Z($\hat{X}_{GEE2}$)	**% Dev***	**AE** Z($\hat{X}_{AE2}$)	**% Dev***
1	1	55,170,060	55,171,168	0.00	55,426,150	0.46	59,275,160	7.44	57,204,800	3.69
	2	54,970,890	54,971,997	0.00	54,972,000	0.00	56,237,960	2.30	57,204,800	4.06
2	1	47,622,320	47,622,320	0.00	47,648,410	0.05	65,359,190	37.24	47,648,410	0.05
	2	22,832,300	22,835,439	0.01	22,848,880	0.07	31,491,840	37.93	22,848,880	0.07

**** % Dev*** stands for the deviation of the cost of solution (in % of the optimal cost)

Table 3.17: Deviation of the Cost of Heuristic Solution from the Optimal Solution for Sample Size of 20% Under the Storage Space Constraint

			Cost of Solution with Real Data		Cost of Solution With Estimated Data Using HRU Heuristic 2					
Lattice Instance	Problem Instance	Optimal Cost $Z(X_{IP2})$	HRU Heuristic 2 $Z(X_{HRU2})$	% Dev*	Shlosser $Z(\hat{X}_{Shl2})$	% Dev*	GEE $Z(\hat{X}_{GEE2})$	% Dev*	AE $Z(\hat{X}_{AE2})$	% Dev*
1	1	55,170,060	55,171,168	0.00	55,426,150	0.46	59,275,160	7.44	57,204,800	3.69
	2	54,970,890	54,971,997	0.00	55,171,170	0.36	58,453,810	6.34	57,204,800	4.06
2	1	47,622,320	47,622,320	0.00	47,648,410	0.05	65,359,190	37.24	47,622,320	0.00
	2	22,832,300	22,835,439	0.01	22,848,880	0.07	38,890,000	70.33	22,839,380	0.03

*** *% Dev*** stands for the deviation of the cost of solution (in % of the optimal cost)

Table 3.18: Deviation of the Cost of Heuristic Solution from the Optimal Solution for Sample Size of 10% Under the Storage Space Constraint

			Cost of Solution with Real Data		Cost of Solution With Estimated Data Using HRU Heuristic 2					
Lattice Instance	Problem Instance	Optimal Cost $Z(X_{IP2})$	HRU Heuristic 2 $Z(X_{HRU2})$	% Dev*	Shlosser $Z(\hat{X}_{Shl2})$	% Dev*	GEE $Z(\hat{X}_{GEE2})$	% Dev*	AE $Z(\hat{X}_{AE2})$	% Dev*
1	1	55,170,060	55,171,168	0.00	55,753,540	1.06	66,295,260	20.17	57,204,800	3.69
	2	54,970,890	54,971,997	0.00	55,263,180	0.53	58,453,810	6.34	57,204,800	4.06
2	1	47,622,320	47,622,320	0.00	56,804,050	19.28	74,527,940	56.50	65,359,190	37.24
	2	22,832,300	22,835,439	0.01	22,848,880	0.07	38,966,380	70.66	31,408,280	37.56

*** *% Dev*** stands for the deviation of the cost of solution (in % of the optimal cost)

Table 3.19: Deviation of the Cost of Heuristic Solution from the Optimal Solution for Sample Size of 5% Under the Storage Space Constraint

Careful review of Tables 3.14 through 3.19 reveals some important observations. In the case of the number of views to be materialized constraint (Tables 3.14 through 3.16), the following points were observed:

1. The HRU Heuristic 1 always produced the optimal solution when applied on the actual data set.

2. In the case of Lattice Instance 1, performance of the Shlosser Estimator and the AE was good. In the case of the Shlosser Estimator, the maximum deviation of the cost of solution from optimal solution was 0.82 for sample sizes 20% and 10%. But for sample size of 5%, for Problem Instance 1, the deviation of the cost of solution from the optimal solution was as high as 18.46%.
3. In the case of Lattice Instance 1, performance of the GEE was poor for Problem Instance 1 and 3. The performance was worse as the sample size went down from 20% to 5%. It did produce the optimal solutions for all sample sizes for Problem Instance 3.
4. The performance of the AE is observed to be the best for all sample sizes. The maximum deviation of the cost of solution from the optimal solution was 4.3% in the case of Problem Instance 3 in Lattice Instance 1 for all sample sizes.
5. In the case of Lattice Instance 2, all the estimators performed pretty well. The only deviation of the solution from the optimal solution was the case of Shlosser Estimator, which was 7.25% for Problem Instance 1 for all sample sizes.
6. One can also notice the improved performance of all the estimators as the number of views to be materialized increased from 5 views to 20 views. The only exception was in the case of the AE for Lattice Instance 1 for all sample sizes where the maximum deviation observed was 4.3%.
7. One interesting point to note is that the performance of the estimators was not highly sensitive to the sample size. Even a sample size of as low as 5% produced acceptable results. Many times, it was not essential to go for sample sizes of 10% and 20%.

In the case of the storage space constraint (Tables 3.17 through 3.19), following points were observed:

1. The HRU Heuristic 2 always produced the optimal solution except in one case when applied to the actual data set. In the case of Lattice Instance 2, for Problem Instance 2, for all the different sample sizes, deviation of the cost of solution from the optimal solution was 0.01, which is very close to the optimal solution.
2. In the case of Lattice Instance 1, the Shlosser Estimator and the AE performed well. For the Shlosser Estimator, maximum deviation of the cost of solution from the optimal solution was 1.06% when the sample size was 5% for Problem Instance 1. For the AE, maximum deviation of the cost of solution from the optimal solution was 4.06% for all sample sizes for Problem Instance 2. Performance of the GEE was comparatively not good. It deteriorated further as the sample size went down. In the case of Problem Instance 1 for a sample size of 5%, the deviation was as high as 20%.
3. In the case of Lattice Instance 2, the Shlosser Estimator and the AE performed well. Their performance was comparatively poor for sample size of 5%. The maximum deviation of the cost of solution from the optimal solution in the case of the Shlosser Estimator was 19% for Problem Instance 1, for the sample size of 5%, and in the case of the AE it was 37.56% for Problem Instance 2, for sample size of 5%. Again the GEE did not perform well. Its performance further deteriorated as the sample size went down from 20% to 5%. For sample size of 5%, the deviation was as high as 70.66%.

4. In general, again, the performance of the estimators was not too sensitive to the sample size except in a few cases. In fact, in the case of sample size of 10%, the AE produced better results than compared to the results obtained from sample size of 20%. In the former case, maximum deviation was 0.03% while in the later case, it was 0.07% for Lattice Instance 2, and for Problem Instance 2. But the deviation was more in the case of sample size of 5% – it was 37% in the case of the AE and 19% in the case of the Shlosser Estimator.
5. If one carefully reviews tables from Table 3.11 through Table 3.13, one can observe that for both lattice instances, even for 60% storage space constraint setting, the system materialized most of the views out of 27 views. The main reason for this could be the root view, which always occupies the maximum space. Almost 45% of the total space in the case of AANS database and 55% of the total space in the case of TPCH database was taken by the root view.
6. If one compares the performance of estimators in the case of the storage space constraint with the performance of estimators in the case of the number of views to be materialized constraint, the deviation of the cost of solution from the optimal solution was always high in the case of the storage space constraint for all settings. One reason for this could be that space constraint puts higher burden on the system when it comes to finding the optimal solution, as the system has to try many more combinations than in the case of the number of views to be materialized constraint. As was discussed in Chapter 2, the 64-node MVS problems with the storage space constraint took almost 90 seconds to find the

optimum solution, while in the case of the number of views to be materialized constraint, it took just 10 seconds to find the optimum solution using LINGO.

3.4.1 Computational Times

The sampling procedures have two major computational components:

1. Drawing the sample from the root view in a given lattice instance.
2. Computing the estimates for the number of rows present in a view in a given lattice instance.

To simplify the computer codes, we drew the sample and populated the sample table on hard disk. We used simple random sampling without replacement method for drawing the samples from the root view. The main purpose of employing statistical sampling to data warehouse is to reduce the computational time needed to count the actual number of records in a table to the minimum possible extent without compromising the accuracy of the solution. One can perform more robust random sampling techniques, but it may proportionately increase the sampling time.

The second part of computational time was computing the estimates from the sample table populated. There were three sub-components in it. First, we need to find the distinct attributes values present in the sample, then compute the frequencies of their occurrence and then apply the three estimators to estimate the number of unique rows present in a view under consideration in a given lattice instance.

The computational times taken by various sampling techniques are shown in Table 3.20. We have collected computational time for both lattice instances, i.e. TPCH and AANS database. In Table 3.20, the second column, named "Actual Number of Rows," shows the total time needed for computing the actual number of rows present in all other views in a given lattice instance from the root view. In the case of Lattice Instance 1, it took 690 seconds in total to count the actual number of rows in all 26 nodes excluding the root view while in the case of Lattice Instance 2, it took 248 seconds. Even though AANS database has more records in the root view (more than nine million) than the number of records in the root view in TPCH database (slightly more than six million), the time taken in the case of AANS database was much less. The main reason for this could be that in AANS database, most key fields are numeric which enables database engine to process the records faster. In columns 4, 5 and 6, we have shown the time needed to compute the Shlosser, the GEE and the AE estimates, respectively.

Lattice Instance	**Computational Time (in Seconds)**				
	Using Actual Number of Rows	**Sample Size**	**Shlosser**	**GEE**	**AE**
1 (TPCH)	690	20%	241.47	221.68	238.26
		10%	120.37	108.64	117.62
		5%	66.28	59.62	63.9
2 (AANS)	248	20%	142.85	114.96	141.01
		10%	73.89	56.51	68.67
		5%	42.54	31.66	37.75

Table 3.20: Computational Times

In the case of Lattice Instance 1, even with a 20% sample size, average time savings were almost 66%, with the GEE taking the minimum time (221.68 seconds). For a sample size of 5%, the savings were as high as 91%, with the GEE again giving the best results (59.62 seconds). But as discussed earlier, the performance of the AE was comparatively best when it comes to estimating the number of rows present in a view in a given lattice instance. In Table 3.20, one can notice that the time taken by the AE is very much comparable with that taken by the GEE.

In the case of Lattice Instance 2, with 20% sample size, time savings in terms of percentage were 42%, 53%, and 43% for the Shlosser, the GEE and the AE estimators respectively. Again, though the GEE took the least time to estimate, its performance in terms of prediction accuracy was not good at all. For the sample size of 5%, the savings were as high as 87% with the GEE and 85% with the AE estimator. As seen earlier, the performance of the AE estimator was best when it comes to prediction accuracy. Based on the above results, the AE has given the best performance in our experimental evaluation both in terms of prediction accuracy and savings in time.

The above statistics clearly indicate the worth of using statistical sampling techniques in generating problem instances for a the data warehouse as it results in significant savings in terms of time without compromising the accuracy in terms of deviation of the cost of solution from the optimal solution. Even a sample size of as low as 5% produced good results. We must also point out that the times taken by the sampling techniques in our experimentation have certain built-in penalties because the necessary data were read from the disk for certain parts of the experimentation. Also as pointed out earlier, most of the DBMS available gather statistics on the table. With appropriate computer programming code, one can use this table statistics and compute the estimations from there. This could definitely result in further savings in terms of time.

In the next section, we briefly discuss the use of statistical sampling techniques and the final results.

3.5 Discussion

Since counting the actual number of unique rows from a very large root node may be time-consuming, the major motivation of our study was to investigate the feasibility of using estimated number of rows using sampling techniques to generate problem instances representative of the actual problem. In our experiment, we use three different sample sizes, viz. 20%, 10% and 5%, to draw the samples from the root view for each Lattice Instance. Then we use two different resource constraints: the first constraint is the number of views to be materialized constraint and second constraint is the storage space constraint. For problems with number of views to be materialized constraints, with these ranges of sample size, all three estimating techniques were able to generate representative problem instances with reasonable accuracy. The deviations between optimal objective function value with actual data and heuristic solution with estimated data were relatively small. In the case of the storage space constraint, the differences in the objective function values using optimal procedure with actual data and heuristic procedure with estimated data were comparatively higher. One reason for this could be that the storage space constraint puts higher burden on the system when it comes to finding the optimal solution as the system has to try all the possible combinations of views that could fit in the available space.

The frequency of occurrence of various attributes or composite attributes was mostly very lowly skewed in both databases. If we consider the sum of absolute deviation proportion of the estimates from the respective actual measures, then we can conclude that the AE sampling method has performed best in both lattice instances for both Problem 1 with equal weights and Problem 2 with equal weights (Table 3.14 through Table 3.19). One can surely conclude that there is a considerable amount of savings in terms of time by employing such statistical sampling techniques to estimate the actual number of rows present in each view in a given lattice structure without compromising the accuracy of the solution (refer to Table 3.20).

In the next section, we provide conclusions and directions for future research.

3.6 Conclusions and Future Research

In this chapter, we have used three different sampling methodologies to generate problem instances representative of the original MVS problem and then we applied the HRU heuristics to solve the problems. The problem instances were generated from two different realistic databases. We have demonstrated that the HRU heuristics perform very well when it is employed with the actual number of rows in each node of the underlying lattice structure. In fact, it produced the optimal solution in almost every problem instance in our investigation.

Subsequently, we applied the appropriate HRU heuristic (HRU Heuristic 1 or HRU Heuristic 2) to solve the MVS problem (Problem 1 with equal weights or Problem 2 with equal weights, respectively) by replacing the actual number of rows present in each view in a given lattice instance with the estimated number of rows. HRU heuristics with the estimated rows provided the optimal solutions in some cases while it failed to find the optimal solutions in others (Table 3.14 through Table 3.19). However, the maximum deviation of the cost of heuristic solution with estimated data from the optimal solution was found to be 71% while the least being 0%. The deviation of 71% was with the GEE estimator which does not perform well in most instances. The maximum deviation with the AE estimator was 38%. Our experimental evaluation shows that the HRU heuristics performed best with the AE estimator. We assume that for a large problem, this loss of accuracy would be acceptable given that the MVS problem is NP-complete (Harinarayan *et al*. 1999).

Whenever an estimated value is used instead of the actual value, certain level of inaccuracies in the solution process is introduced. However, the success of using estimates in solving the MVS problem may depend on the internal characteristics of the MVS problem itself. In certain lattices, introduction of minor variations in the values of the parameters may not make significant differences. Further research is needed to identify the sensitivity of a solution for a given MVS problem or for a given class of the MVS problems.

In this chapter we have demonstrated that employing sampling methodologies can provide acceptable solution to the MVS problem. Obviously, further research is needed to explore more refined data structures so that the sample table can be rapidly constructed inside the memory instead of writing it in a secondary storage device.

CHAPTER 4: SIMULATION MODEL AND ANALYSIS OF A DATA WAREHOUSE

4.1 Introduction

A data warehouse can be defined as a database of databases. It brings together a wide range of information from many different heterogeneous independent data sources into one centralized data repository to support business analysis activities and decision-making tasks. In Chapter 2 we presented two heuristic procedures for solving a special case of the MVS problem. We also presented 0-1 Integer Programming models to address four different versions of the MVS problems. In Chapter 3, we explored the use of statistical sampling techniques to estimate the number of rows present in each view in a given lattice structure and examined the difference in the objective function values between heuristic solutions to estimated problem instances and the optimal solutions to corresponding actual problem instances.

In this chapter, we explore the application of a systems dynamics approach and simulation to model a data warehouse and estimate its performance. Data warehouses are large complex systems with many interacting non-linear components. A small change in any one component may have a dramatic impact elsewhere in the system. Consequently the behavior of data warehouse is often unpredictable (Hillard *et al.* 1999). Data warehouse managers may find a simulation model useful in identifying critical components, diagnosing problems and optimizing the overall design. It will allow users to pinpoint performance issues or troubleshoot performance problems that may have occurred hours or days earlier.

The next section briefly describes the systems dynamics approach to data warehouse modeling. In Section 4.3, we briefly review the related literature. Section 4.4 describes the conceptual framework of the simulation model. In Section 4.5, we discuss the simulation model developed using the ARENA simulation package along with its results. In Section 4.6, we discuss the experimental design for conducting experiments in brief. Experimental results are discussed in Section 4.7. In Section 4.8, we conclude this chapter and discuss future research to be pursued.

4.2 Systems Dynamics Approach to Data Warehouse

Data warehouses are large complex systems with many non-linear interacting components (Hillard *et al.* 1999). Many different technologies are needed to assemble an effective data warehouse. For example, one needs Extract-Transform-Load (ETL) tools that will fetch the data from multiple heterogeneous data sources spread across geographical boundaries. These tools help to extract information, transform information from different domains into one standard domain, and then load that information into the data warehouse. These tools are very complex in nature. Secondly since the data is spread across different locations, one often needs a high speed network that will support fast data communication between the data sources and the data warehouse. Hence network bandwidth plays a crucial role in determining the response time faced by data warehouse users.

Once the information is loaded into the data warehouse, one needs to materialize a set of views to improve query performance. This process, as seen in the last two chapters, is very involved and time intensive. Once the views are materialized, they need to be maintained to reflect the latest information, as the information at the data sources keep changing over time. Maintaining views is again a very time-consuming process. Many researchers have addressed this issue (Colby *et al.* 1997, Gupta 1999, Kalnis 2002, Mumick *et al.* 1997).

Once the data warehouse is properly set up, the data warehouse users need various query-and-reporting tools, multidimensional analysis tools, and statistical tools to mine the data warehouse for valuable information. There are several tools available in the market, such as Actuate's Enterprise Reporting Applications (http://www.actuate.com), Panorama's Business Intelligence Platform (http://www.panoramasoftware.com), Microsoft OLAP tools, Oracle Discoverer, and Oracle Reports. Though these tools are powerful, they are seldom comprehensive. Often companies need to customize them to suit their requirements. This can be proved to be a very tedious task. Furthermore, data warehouse end users may be spread over geographically diverse locations, which may necessitate the use of high speed network to support faster data communication. Hence network bandwidth may again become critical a factor influencing the response time experienced by the end users.

Last but not the least, data warehouse performance also depends on the capacity and speed of the disk farm, the indexing and partitioning strategy used to optimize the database efficiency, the CPU capacity, and the RDBMS engine.

In summary, the performance of a data warehouse is difficult to predict because: (i) the data warehouse is comprised of a number of different and interacting components, (ii) the data warehouse is dependent on other systems for its data and (iii) the usage that will be made of the system by end-users is often difficult to predict.

In today's decision support environment where time is a very critical factor, the data warehouse manager must make sure that end-users are getting reliable answers to their questions within a reasonable time. One method of ensuring timeliness and reliability is to experiment with real system components. Experimenting with real components is a risky venture as a small change may has profound impact on system performance.

When dealing with such a complex system, a simulation model could be a viable approach to predict the behavior and performance of a data warehouse system. The simulation approach to data warehouse performance applies the principles of *systems dynamics*, used in real-world simulation applications such as biological, engineering, and nuclear research. It can be used to simulate the behavior and performance of the data warehouse system based on its overall design. Such a simulation approach can have a number of benefits:

i. Data warehouse managers can improve the performance of their systems. They can walk through various scenarios and as a result can configure their new or updated systems to be more reliable and efficient.

ii. Data warehouse managers can manage risk better and plan better capacity utilization. They can justify or negate investment decisions and can accurately predict when they need to update, add or replace system parts.

iii. Data warehouse managers will have more complete understanding of the interrelationships between components in an existing data warehouse configuration, and how multiple factors influence the data warehouse performance. This provides an ability to understand both opportunities for synergies within the design and areas of inefficiencies.

In the next section, we briefly discuss one related research work in this area.

4.3 Related Work

Considering that data warehouse technology itself is a budding technology, there is not much published research with regard to simulating the behavior and performance of data warehouse systems. One study by Robert Hillard, Peter Blecher and Peter O'Donnell (1999) applies the concept of Chaos Theory to the operation of data warehouses. First a brief explanation of chaos theory follows.

Chaos theory is used to understand and make predictions about apparently random behavior of complex systems that have many interacting non-linear components. The key aspect of chaotic systems is that they are very sensitive to even small changes in initial conditions (Gleick 1987, Medio 1992). This means that nearly identical systems, with only slight differences, may behave very differently.

Hillard *et al.* (1999) argued that data warehouses are large complex systems with many non-linear interacting elements. This creates a tendency for periods of chaotic behavior. The application of chaos theory will help in developing an understanding of the variability of data warehouse performance commonly experienced by the developers and managers of data warehouses. They further argued that chaos theory could be used to minimize the impact of such erratic performance on the users of the system.

In the next section, we briefly discuss the conceptual framework for simulating the behavior and performance of a data warehouse system.

4.4 Conceptual Framework for Simulating Data Warehouse Operation

In general the operational performance of a data warehouse depends on the interaction of various parameters, like the length of time required for the uploading cycle, the types of ad-hoc queries posed to data warehouse, the frequency of these queries, the mean time between the failures of the equipment, user behavior and the performance of the database management system itself. The network of influences between and within these parameters is very complex. We have developed a conceptual framework of the simulation model for simulating the behavior and performance of a data warehouse. The details of this model are shown in Figure 4.1. The major components of the conceptual model are:

a. Design Criteria

Performance of a data warehouse largely depends on the number of views materialized. Given a set of materialized views, if a query is answered directly from the materialized views, it will be executed quickly with minimum response delay. On the other hand, there might be queries which could not be directly answered from the materialized views, but could be answered indirectly using one or more of the materialized views. Such queries might take more time to execute compared to the former ones. Some queries may need to refer back to the root view (base cuboid), which will always be materialized and which has the lowest level of summarization with all possible dimensions and levels in those dimensions. In this case the query would likely to take a long time to execute. Therefore, we have various categories of business queries that are directly related to the views materialized in the data warehouse.

b. View Materialization Policy

There are two view materialization policies possible. The first policy is the *fixed view materialization policy*, which uses a model to decide the set of views to be materialized and once such views are materialized they are never changed. Such views are updated periodically to reflect the changes in the source information, so that the end users always get current data. The second policy is the *dynamic view materialization policy*, which assumes that over time, business requirements change, resulting in a change in the query pattern. This could lead to *dynamically* changing the views to be materialized (Kalnis 2002). In this work, we have focused on the *fixed view materialization policy*. However, from a simulation perspective, this is not a serious limitation, since our query arrival behavior captures both types of queries. The only limitation may be that we are not accounting for query processing time lost due to change in the number or sets of views materialized.

c. Environmental Variables

Certain environmental variables have profound effects on the performance of a data warehouse. One such environmental variable is equipment failure. There could be many causes of equipment failure including power failure, server crashing and computer hardware failure like hard-disk or RAM or router. When such instances occur, data warehouse systems fail and all the queries in the system are lost.

d. View Maintenance Practices

The materialized views need to be maintained so as to reflect changes in the source information. The data may be loaded into the system on a monthly, weekly or daily basis depending on the organization’s requirements. During the start of the uploading cycle queries already being processed are generally allowed to execute to completion; but waiting queries are lost. No new queries are accepted during the uploading process.

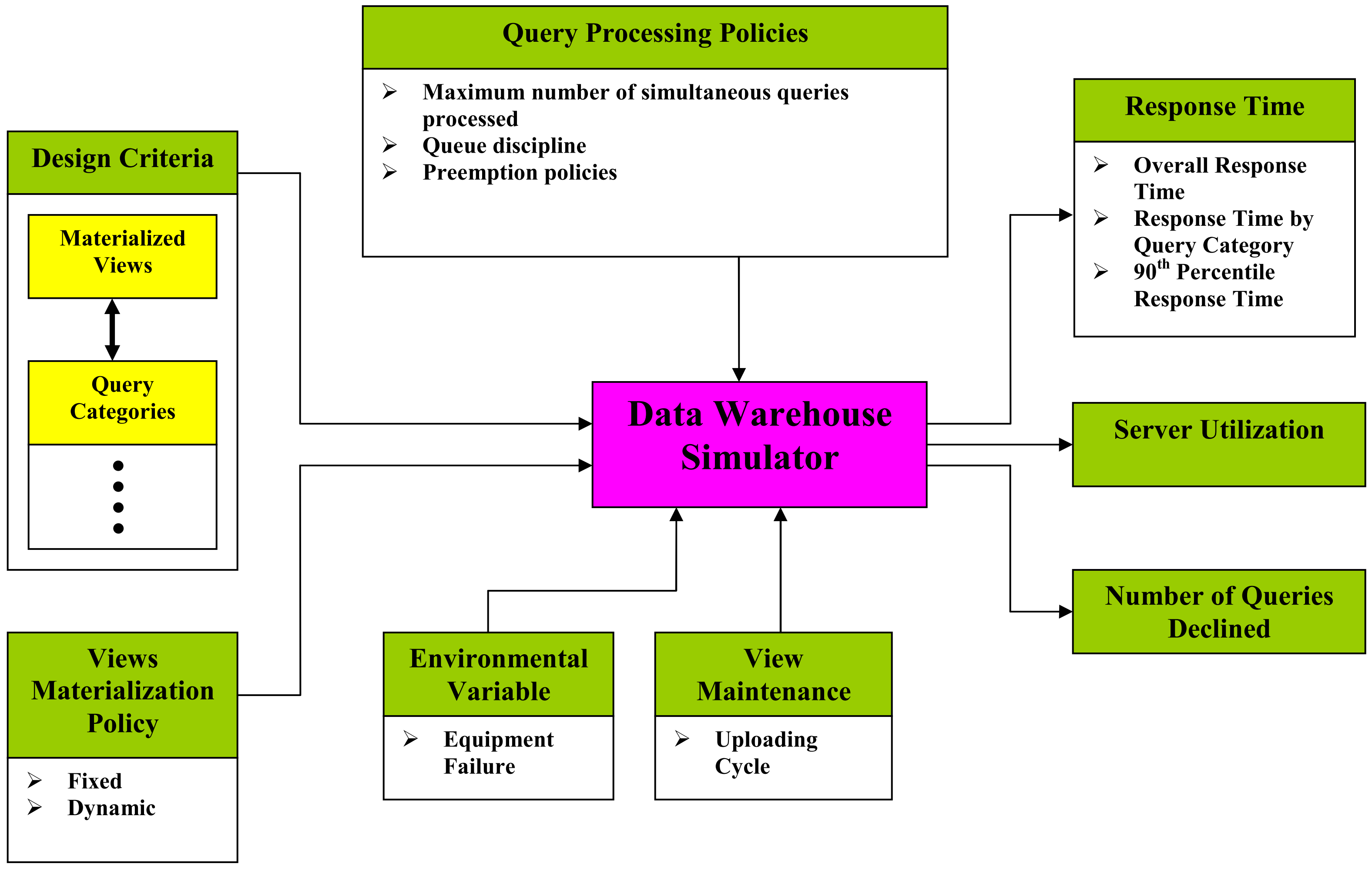

Figure 4.1: Conceptual Framework for Simulating Data Warehouse Performance

e. Query Processing Policies

If only one query is being processed at the data warehouse, it will be faster since all the resources (CPU, RAM, etc.) are devoted to that single query. If more than one query is being processed simultaneously, the processing of all queries will be slowed because of the sharing of resources.

In the next section, we briefly discuss and provide a model for implementation of the conceptual framework discussed in this section using ARENA simulation package.

4.5 Implementation of Conceptual Framework in ARENA

In the previous section, we discussed the conceptual framework of the simulation model for simulating the behavior and performance of a data warehouse. In this section, we present the implementation of the conceptual framework of the simulation model using ARENA simulation package. Our objective is to predict the performance of a data warehouse under varying design conditions and query processing policies in terms of the average time to process a query, the average time to process each category of query, the server utilization and the number of queries declined.

As discussed in Chapter 3, we have populated a 1 GB data warehouse from the Transaction Processing Council (www.tpc.org) website. The Transaction Processing Council website provides the data to populate big experimental data warehouses (ranging from 1 to 1000 GB) and has been widely used by industries for benchmarking purposes.

Running this data warehouse on our computer system, we gained some insight into the parameters relevant for simulation model building such as query processing time, delays experienced by queries because of simultaneous processing, and the relationship between these parameters and the views materialized in this data warehouse.

Figure 4.2 shows the main features of the simulation model developed using ARENA simulation package. The major components of the ARENA simulation model are discussed below:

a. Design Criteria Based on Materialized Views: Query Categories

This block generates queries that the users pose to the data warehouse. There are two *create* modules. The first generates queries (entities) that could be directly answered from the materialized views. The second generates queries that could not be directly answered from the materialized views. The inter-arrival rate is assumed to be exponentially distributed and the processing time is assumed to be normally distributed for both query types. The specific parameter values used are presented in Table 4.1. The type of distribution used for the inter-arrival time and service was broadly justified from general queuing and database literature as well as our experience with the TCPH and AANS database.

b. Decision Block and Waiting Queue

As mentioned earlier, the server in this case can handle at most three queries simultaneously. Details of this construct and the justification are given under query processing policies. This is modeled using three process modules in ARENA. Each incoming query checks the availability of one of the three process modules. If three queries are already being processed, then all arriving queries wait in a queue. As soon as the server becomes available, queries in the queue may proceed for processing.

c. Data Warehouse Simulator with Query Processing Policies

The data warehouse simulator is the engine that processes the incoming queries based on their critical attribute values. Modeling the processor of the data warehouse was somewhat difficult since a typical computer can process several queries at a time while a processor construct in ARENA can handle only one query at a time. We decided to allow at most three queries to be processed simultaneously. We do this by first creating three process modules, and then activating them one at a time as the number of queries in the system goes from 0 to 3, after which the number of active process modules stays at three satisfying the condition of processing a maximum of three simultaneous queries. Finally, the processing time of each query gets increased as the number of simultaneous queries in process increases. The following describes the way query processing takes place in the simulator:

i. As the number of simultaneous queries under process increases, the processing time of all queries under process increases. This approximately mimics the simultaneous processing of queries by a real CPU. In this version of the simulation model, only three queries are allowed to be processed simultaneously. In general, the higher the number of simultaneous queries, the higher the delay in response will be. Queries arriving while all three process modules are busy will be held in the queue until one of the process modules becomes free.

ii. The number of queries being processed simultaneously determines the appropriate delay factor to be applied to the remaining processing time of all queries under process. This is done each time a process module begins or completes a query. The delay factors used in our simulation are given in Table 4.1. The function of the delay factor is simply to linearly increase or decrease the remaining processing time as the number of queries in process changes by 1. For example, at simulation time *t*, if the system is currently processing two simultaneous Type 1 queries (which implies that there are no waiting queries) with remaining processing time t_1 and t2, and if a third Type 2 query arrives with a simulated processing time of t3, then using the delay factors given in Table 4.1, the new processing time of the three queries will be $(1.5/1.25)^* t_1$, $(1.5/1.25)^* t_2$ and 2^*t_3, respectively. Suppose at time t', processing of Type 2 query is completed, that is t_2 becomes 0, and no new query arrives (note that waiting line must be empty, since only two process modules are active at time t'), the new processing time for Type 1 query will be $(1/1.25)^*t_1$. That is the factors given in Table 4.1 applies to the original simulated time and since some of the original times have already been increased when the number of simultaneous queries processed increased from 1 to 2, it must now be adjusted suitably when it goes from 2 to 3. All of this is achieved by first preempting the current queries under process, changing the remaining processing time and resuming the processing by the respective servers from the point of preemption.

iii. At the time of arrival of the uploading task, queries already in-process will be processed. But the queries waiting in the queue, if any, will be lost. Queries that arrive after the arrival of the uploading task will also be lost. Once uploading is complete, normal query processing will resume.

iv. During the failure of equipment, all incoming queries as well as queries in-process or waiting at that instance will be lost.

v. If the system fails during the uploading cycle, uploading will fail and the system will revert back to earlier status. The time required to put the system back to work will include the time needed for the uploading task as well, so that data warehouse users access the latest information. As soon as the system is up, the incoming queries will follow the previous mentioned rules.

d. View Maintenance Practices: Uploading Cycle

Here, this is modeled by a single query that seizes the server for a certain period of time, thus simulating the time required for loading the latest information from the data sources to the data warehouse. The uploading cycle query first checks if there are any queries that have already seized the server. In that case, the uploading cycle query will wait in the queue until the queries that seize the server are released out of the system. At this instance all the waiting queries if any will be lost. Then the uploading query will seize the server for certain specified time and during this time, all the incoming queries will be lost. As soon as the uploading is complete, the incoming queries will be allowed to enter the server as per the earlier mentioned policies.

e. Environmental Variable: System Failure

Here, this is modeled by a single query that enters the server and seizes it for a specified period of time. When the system failure occurs, all the queries in the system as well as those waiting in the queue are lost. Incoming queries are also disposed off immediately. If update is going on during the time of failure, it will fail and data warehouse will be reverted back to its original state. As soon as the failure query leaves the systems, the server will be up and the incoming queries will be entertained in the server as per previously mentioned rules.

f. Disposing Policies

The departing query has to perform a check on the server before being disposed off. This accomplishes the delaying or speeding up of the queries under process, if any, based on certain pre-emption policies (refer to Section 4.5.c.ii).

In the next section, we discuss the experimental results obtained for a simulated run of one week.

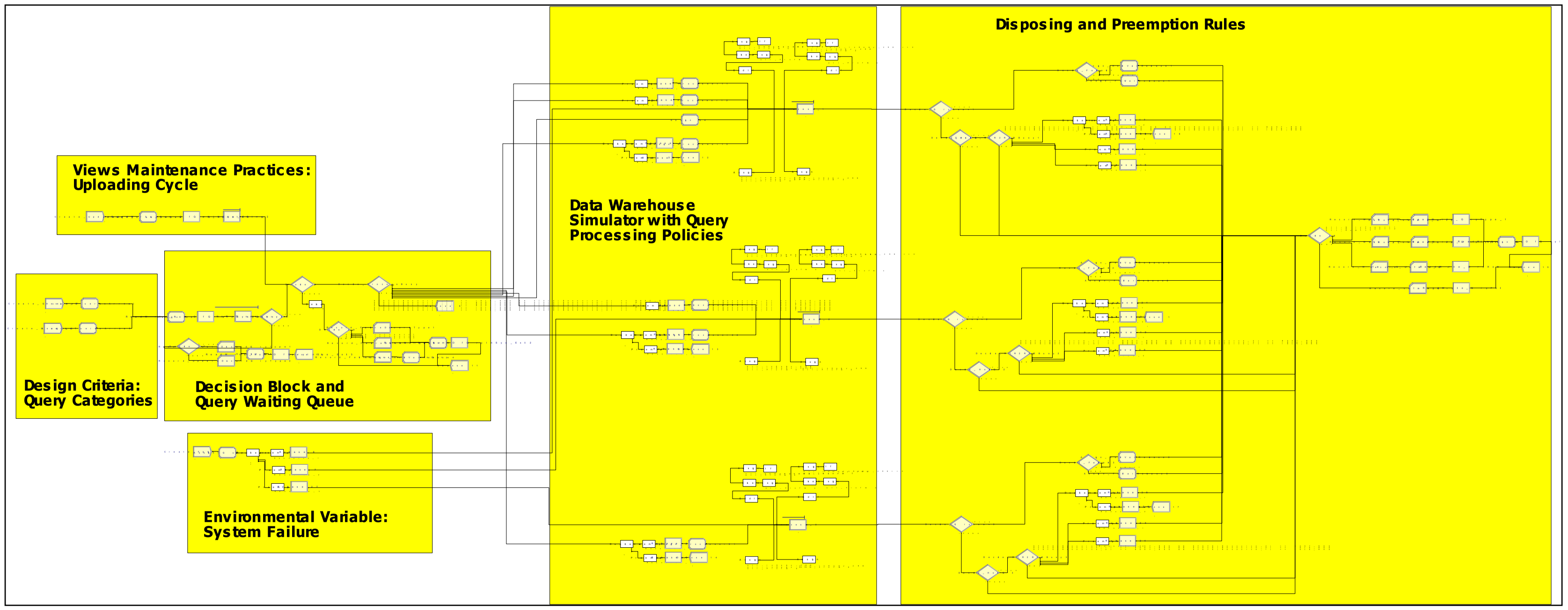
Views Maintenance Practices: Uploading Cycle
Design Criteria: Query Categories
Decision Block and Query Waiting Queue
Environmental Variable: System Failure
Data Warehouse Simulator with Query Processing Policies
Disposing and Preemption Rules

Figure 4.2: Implementation of Conceptual Framework of Simulation Model in ARENA

4.6 Experimental Design

The simulation experiments were designed to demonstrate the usefulness of simulation model in a data warehouse design context. For each experimental setting, model was run for a simulated period of eight days. Data for the first day was discarded to let the system reach steady state conditions. Table 4.1 shows various model input parameters with corresponding values.

Sr. No.	Query Type	Arrival Time Distribution (in minutes)	Processing Time Distribution (in minutes)	Delay Factors		
				No Delay	First Delay	Second Delay
1	Query Type 1	EXPO(x)*	NORM(10,2)	1	1.25	1.5
2	Query Type 2	EXPO(y)*	NORM(20,3)	1	1.5	2
3	Failure	XPO(7200)	EXPO(500)	–	–	–
4	Update	CONSTANT 1440	CONSTANT 60	–	–	–

* "x" and "y" are specified in Table 4.2

Table 4.1: Experimental Parameters

In many service oriented simulation models, we hypothesize a Poisson arrival process, which leads to exponentially distributed inter-arrival times. To reduce variance of the difference between outcomes (such as average waiting time, average turnaround time, etc.) of different experimental settings, we used common random number generators for various random processes in the model. Processing times for the queries are assumed to come from the Normal distribution with a mean of 10 and 20 minutes, and standard deviation of 2 and 3, respectively. As mentioned earlier, these were estimates derived on the basis of our experiments running the TPCH and AANS databases in several servers. Table 4.1 also shows the Delay Factors used for each type of query. The values for First Delay and Second Delay are 1.25 and 1.5, respectively, for Query Type 1, and 1.5 and 2, respectively, for Query Type 2. First Delay occurs when two queries are being processed by the server simultaneously. Second Delay occurs when three queries are being processed by the server simultaneously. Please refer to section 4.5.c.ii for details.

The time between failures is assumed to come from the Exponentially Distribution with a mean value of 7200 minutes. The time to repair a failure is also assumed to be exponentially distributed with a mean of 500 minutes. Data uploading is assumed to take place daily starting at mid-night. This process is assumed to take 60 minutes.

Sr. No.	**Experimental Setting**	**Inter Arrival Time (minutes)**		**Query Mix (y/x)**
		Query Type 1 (x)	**Query Type 2 (y)**	
		[Proc = NORM(10,2)]	**[Proc = NORM(20,3)]**	
1	a	20	50	2.50
2	b	15	30	2.00
3	c	16	44	2.75
4	d	15	40	2.67
5	e	14	36	2.57
6	f	13	32	2.46
7	g	12	28	2.33
8	h	11	24	2.18
9	i	10	20	2.00

Table 4.2: Experimental Settings

To achieve a variety of operating conditions, we kept the processing time parameters unchanged while the inter-arrival rate was varied. Table 4.2 shows the nine different experimental settings used in our experiment. For example, in setting "a," the mean inter-arrival time for Query Type 1 is assumed to be 20 minutes and for Query Type 2, it is assumed to be 50 minutes. Again all inter-arrival times follow exponential distribution. Since Query Type 2 queries cannot be directly answered from the materialized views, they have a longer processing time and they are also presumed to occur less frequently since they system is hopefully designed with appropriate set of views materialized. With the query mix combination and mean arrival rate, we maintain a server utilization of 75% to almost 99%. The query mix is selected to be between 2.00 and 2.75.

In traditional queuing analysis, server utilization is given by λ/μ for single server systems and $(\lambda/s)*\mu$ for a multi server system with s servers, λ is the arrival rate and μ is the service rate. The system in this simulation is little tricky in that the number of active servers changes from 1 to 3 and the service time and hence the service rate of all servers changes as the number of active servers changes. As described earlier, this helps us mimic the processor of a data warehouse server more accurately. We calculated the approximate utilization rate by judiciously combining simulated probability of system having 0, 1, 2, 3 or more queries in the system, types of queries in the system, and the corresponding rates at which each active server is working. For example, if there are two Type 1 queries in the system, and the experimental setting is "*b*," inter arrival time of Query Type 1 is Expo(15), Query Type 2 is Expo(30), and service time is Normal (10,2) for Query Type 1 and Normal(20,2) for Query Type 2. The delay factors will make the simulated remaining service time of the queries in process and/or the new arrival to go to 1.25*(whatever was remaining from the original service time at the time of arrival of the second query). This will help us estimate the instantaneous service rate of the system when the system has two Type I queries. There are many combinations of system states each of which will have its own instantaneous service rate as calculated above. Weighted average of these service rates will be used as an estimate for the average service rate. The average inter-arrival rate is known for setting "*b*". Since the service rate already incorporates the expected service time for each type of query and calculates its rate using that, the arrival rate used is simply the sum of the two arrival rates. Ratio of the arrival to service rates gives the estimated utilization. The system gets emptied everyday at 12.00 midnight to process the update query. The simulated system is thus a series of one day simulations. This midnight emptying process also allows us to slightly overload the system and still not experience total clogging of the system, which would have happened in settings "*h*" and "*i*".

4.7 Experimental Results

Table 4.3, Table 4.4 and Table 4.5 present the system performance obtained from our model for all the experimental combinations and each query type. For each experimental setting, Tables 4.3 through 4.5 show the simulated server utilization, the average query waiting time, the average query processing time, the average query turnaround time and the 90th percentile of the query turnaround time. As expected, we observe that as the mean inter-arrival time decreases in the subsequent experimental setting from “a” through “i” the simulated server utilization, the waiting time, the average turnaround time and the 90th percentile turnaround time increase. Mean processing time increases modestly. This increase is a result of the inherent delay in simultaneous processing of queries as modeled here by the use of delay factors. As sever utilization increases, larger values of the delay factor are applied more frequently thereby increasing the time to process queries.

Experimental Settings	**Simulated Server Utilization**	**Query Type 1**			
		Average Waiting Time	**Average Processing Time**	**Average Turnaround time**	**90 percentile of Turnaround Time**
a	0.75	2.55	12.48	15.03	23.45
b	0.89	7.67	13.59	21.26	39.35
c	0.84	3.93	12.87	16.80	26.91
d	0.85	5.98	13.17	19.15	36.17
e	0.89	6.10	13.43	19.54	35.32
f	0.91	13.39	13.86	27.24	59.84
g	0.91	17.90	14.28	32.17	57.18
h	0.91	42.83	14.52	57.34	147.70
i	0.99	90.87	14.71	105.60	188.10

Table 4.3: Query Type 1 Statistics

Experimental Settings	Simulated Server Utilization	Query Type 2			
		Average Waiting Time	Average Processing Time	Average Turnaround time	90 percentile of Turnaround Time
a	0.75	2.77	31.18	33.95	47.96
b	0.89	8.69	34.58	43.27	70.99
c	0.84	5.10	32.78	37.88	59.67
d	0.85	6.01	34.22	40.24	60.13
e	0.89	6.95	34.24	41.19	62.69
f	0.91	13.33	34.40	47.73	85.91
g	0.91	17.94	36.26	54.20	85.92
h	0.91	40.14	38.35	78.49	146.30
i	0.99	93.69	39.70	133.40	221.30

Table 4.4: Query Type 2 Statistics

Experimental Settings	Simulated Server Utilization	Overall			
		Average Waiting Time	Average Processing Time	Average Turnaround time	90 percentile of Turnaround Time
a	0.75	2.61	17.80	20.41	30.42
b	0.89	7.98	20.02	28.00	49.04
c	0.84	4.23	18.00	22.24	35.35
d	0.85	5.99	18.58	24.57	42.33
e	0.89	6.33	19.01	25.34	42.66
f	0.91	13.37	19.50	32.86	67.00
g	0.91	17.91	20.28	38.19	65.04
h	0.91	42.04	21.51	63.55	147.32
i	0.99	91.76	22.65	114.42	198.64

Table 4.5: Overall Statistics

Figures 4.3 and 4.4 represent parts of these data as bar charts. They show the comparison between the average turnaround time and the 90th percentile turnaround time respectively for Query Type 1, Query Type 2 and Overall for experimental settings of “a” through “i” along with the corresponding simulated server utilization. One can see that as the value of inter-arrival time decreases the average turnaround time and the 90th percentile turnaround time increases. As one can notice from these figures, after experimental setting “f” the performance of system deteriorates considerably. A data warehouse manager could use such information to plan future expansion of the system to improve and protect performance levels.

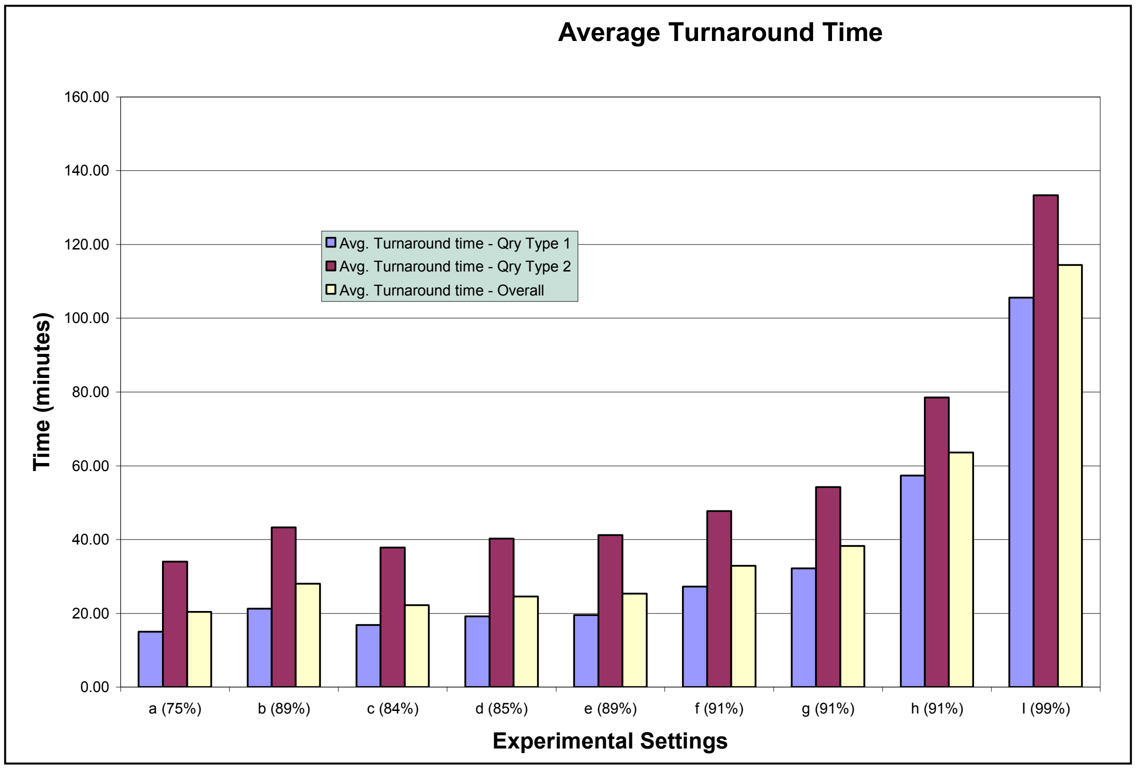

Figure 4.3: Bar Chart Showing Comparison of Average Turnaround Time Between Query Type 1, Query Type 2 and Overall

Table 4.6 shows the number of queries discarded due to data warehouse update and data warehouse failure for Query Type 1, Query Type 2 and the total number of queries discarded. Again one can notice the increase in the number of queries discarded as the query inter-arrival time decreases. Since update takes place every day, this has direct effect on the number of queries discarded during that time. But queries discarded due to data warehouse failure do not seem to be affected considerably.

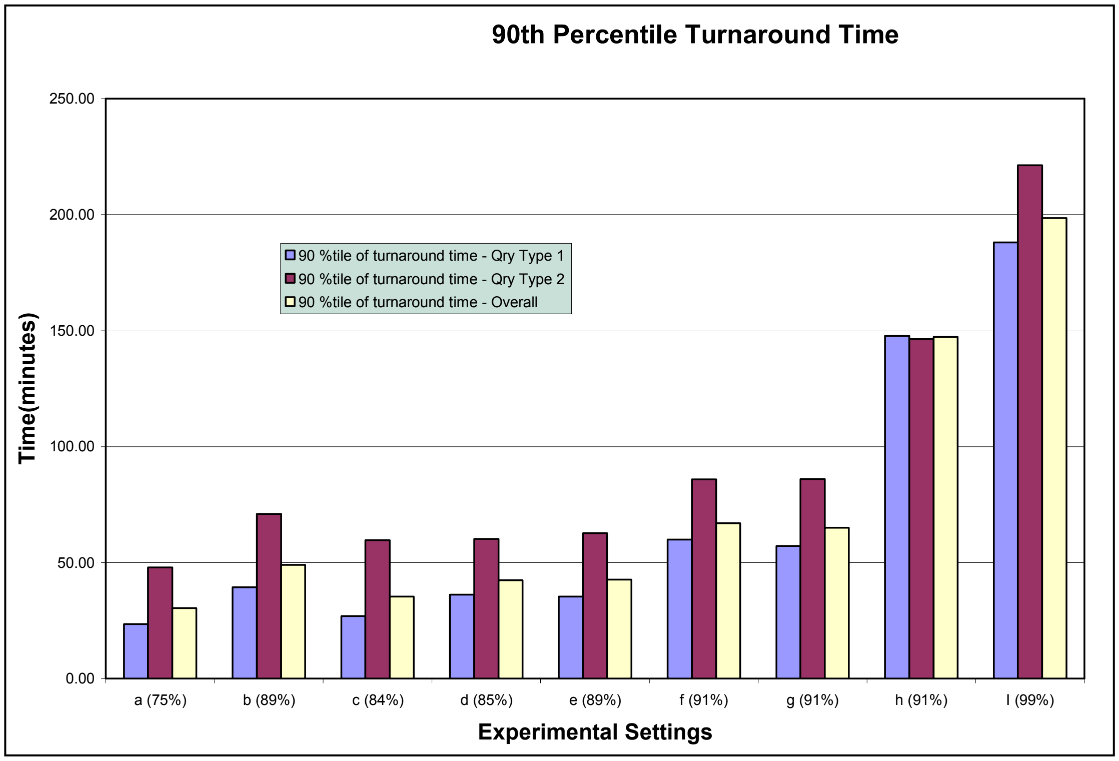

Figure 4.4: Bar Chart Showing Comparison of 90th Percentile Turnaround Time Between Query Type 1, Query Type 2 and Overall

Sr. No.	Experimental Settings	Queries Discarded due to Data Warehouse Update		Queries Discarded due to Data Warehouse Failure		Total Queries Discarded
		Query Type 1	Query Type 2	Query Type 1	Query Type 2	
1	A	33	7	4	1	45
2	B	36	19	3	3	61
3	C	37	10	5	0	52
4	D	37	10	6	2	55
5	E	44	9	0	5	58
6	F	52	12	3	4	71
7	G	66	27	4	3	100
8	H	86	36	11	3	136
9	I	173	78	32	13	296

Table 4.6: Number of Queries Discarded

In the next section, we conclude this chapter and provide directions for future research.

4.8 Conclusions and Future Research

Data warehouses are large complex systems with many different interacting components. A small change in any one component may produce dramatic changes somewhere else in the system. As such a data warehouse environment is very dynamic and sensitive and can exhibit periodic chaotic behavior. In this chapter, we presented a conceptual framework for simulating the operation of data warehouses. We have also developed a simulation model that will predict the behavior and performance of a data warehouse system given initial values for some parameters. We used the TPCH benchmark database to obtain insights into some critical parameters like queries processing times, effect of simultaneous processing on queries processing time, and their relationship to the views materialized.

The simulation model is built using the ARENA simulation package. In this simulation model, we have provided the provisions for different query categories with different processing times, effect of equipment failure on the system, and uploading cycle to upload the current information from the data sources to make the model more realistic. We have also provided the provisions for delaying and speeding up of the query processing depending upon the number of simultaneous query occupying the data warehouse processor.

Our experimental evaluation points to the fact that a simulation model can be used to model a data warehouse environment, thus providing a justification for the technical feasibility. Data warehouse managers can use such models to enhance the performance reliability of the data warehouses. Our experimentation also shows that different query types (those which could be directly answered from the materialized views and those which could not be directly answered from the materialized views) have significant impact on the system’s performance. Future research could focus on validating the model against some real-world data warehouses. Also one can extend the simulation model discussed here to incorporate more features making the model more realistic.

CHAPTER 5: CONCLUSIONS AND FUTURE RESEARCH

Data warehouses are seen as strategic weapon to gain competitive advantage for businesses. A data warehouse extracts, integrates, and stores the “relevant” information from multiple, independent, and heterogeneous data sources into one centralized data repository to support the decision making information needs of knowledge workers and decision makers in the form of Online Analytical Processing (OLAP). Business analysts then run complex business queries over the data stored at the data warehouse to mine the valuable information and identify hidden business trends. Results of such queries are generally pre-computed and stored ahead of time at the data warehouse in the form of *materialized views.* This drastically reduces the query execution time to minutes or seconds which otherwise may take hours or even days to complete.

There are many architectural issues involved concerning the design of a data warehouse. In this dissertation, we have concentrated on three such issues. The first issue concerns the selection of views to be materialized. Selecting the right set of views to materialize is a non-trivial task. This problem has been shown to be NP-Complete in data warehouse literature (Harinarayan *et al*. 1999). We have developed two heuristic procedures to solve two different versions of the MVS problem. We have also developed the 0-1 Integer Programming models to find the optimum solutions to four different versions of the MVS problem. We compared the results obtained by using the heuristic procedures with that obtained by using the 0-1 Integer Programming models and reported the findings. Future research in this area could focus on validating these heuristic procedures against some real-world data warehouse systems. One could also focus on performing sensitivity analysis of weights associated with materialized views and their relationship to optimal and/or heuristic solutions.

The second issue deals with employing statistical sampling procedures to generate problems instances given a huge root view (least aggregated data view) and an associated lattice structure. Most of the heuristic procedures available in the data warehouse literature, including ours, assume that the number of rows present in each view in a given lattice structure is known ahead of time. But actual counting of the number of rows present in each view takes considerable time. Such delays are not permissible in today's decision support environment. We have explored the use of statistical sampling techniques to address this issue. We have applied three well know estimators from the database sampling literatures to two realistic data warehouses. The estimators used are the Shlosser Estimator (Hass *et al*. 1995, and Shlosser 1981), the Guaranteed Error Estimator (Charikar *et al.* 2000, Chaudhuri *et al*. 1998) and the Adaptive Estimator (Charikar *et al.* 2000). Then we have employed the 0-1 Integer Programming model as well as the heuristic procedures developed by Harinarayan *et al.* (1999) to the actual as well as to the estimated data and compared the results. Our findings suggest substantial computational time savings without compromising the accuracy of the solution. Future research in this area could focus on performing the sensitivity analysis of a solution for a given class of the MVS problems and developing more refined data structures to store sampled tables in the computer memory to further reduce the computational time. One can also address the confidence interval issues associated with the difference between the objective function values of estimated and actual problems.

The third issue deals with simulating the behavior and performance of a data warehouse system based on its overall design. A data warehouse system is a large complex system with many interacting non-linear components. The data warehouse environment is said to be very dynamic and sensitive and exhibit periodic chaotic behavior. We have provided a conceptual framework for simulating the operation of data warehouses. We have also developed a simulation model using ARENA simulation package that will simulate the behavior and performance of the data warehouse system given initial values for some system parameters. Our experimentation shows that simulation can be used to construct a realistic model of a data warehouse, thus providing justification for the technical feasibility. Future research in this area could focus on validating this model against some real-world data warehouse systems. One can also focus on extending this model and incorporating more features to make the model more realistic.

BIBLIOGRAPHY

1. Adiba, M. E. & Lindsay, B. (1980). Database Snapshots, Proceedings of the Sixth International Conference on Very Large Databases, Montreal, Canada, pp. 86-91.

2. Blakeley, J. A., & Martin, N. L. (1990). Join Index, Materialized View, and Hybrid Hash Join: A Performance Analysis, Proceedings of the 6th IEEE Conference on Data Engineering, Los Angeles, CA, pp. 256-263.

3. Bunge, J. and Fitzpatrick M. (1993). Estimating the Number of Species: A Review. Journal of the American Statistical Association 88, pp. 364-373.

4. Burnham, K. P. & Overton, W. S. (1978). Estimation of the Size of a Closed Population When Capture Probabilities Vary Among Animals. Biometrica, 65(3), pp. 625-33.

5. Burnham, K. P. & Overton, W. S. (1979). Robust Estimation of Population Size When Capture Probabilities Vary Among Animals. Ecology, 60(5), pp. 927-936.

6. Charikar, M., Chaudhuri, S., Motwani, M. & Narasayya, V. (2000). Towards Estimation Error Guarantees for Distinct Values. Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Dallas, TX, pp. 268-279.

7. Chaudhuri, S., Motwani, R., & Narasayya, V. (1998). Using Random Sampling for Histogram Construction. Proceedings of the 1998 ACM SIGMOD Conference on Management of Data, pp. 436-447.

8. Chaudhuri, S. and Shim, K. Including Group-by in Query Optimization. (1994). Proceedings of the 20th International Conference on Vary Large Databases, Santiago, Chile, pp. 354-366.

9. Colby, L., Kawaguchi, A., Lieumen, D., Mumick, I. and Ross, K. (1997). Supporting Multiple View Maintenance Policies. Proceedings of ACM SIGMOD 1997, International Conference on Management of Data, pp. 405-416.

10. England, Ken (2001). Microsoft SQL Server 2000 – Performance, Optimization and Tuning Handbook. Digital Press.

11. Gleick, J. (1987). Chaos: Making a New Science. Sphere Books: London, UK.

12. Gray, J., Chaudhuri, S., Bosworth, A., Layman, D., Reichart, D., & Venkatrao, M. (1997). Data Cube: A Relational Aggregation Operator Generalizing Group-by, Cross-Tab, and Sub-Totals. Data Mining and Knowledge Discovery, 1, pp. 29-54.

13. Gray, P. and Watson, H. J. (1998). Decision Support in the Data Warehouse. Data Warehousing Institute Series from Prentice Hall.

14. Grover, R. (1998). Identification of Factors Affecting the Implementation of Data Warehouse. Ph.D. dissertation, School of Management, Auburn University Graduate School.

15. Gupta, A., Harinarayan, V. and Quass, D. (1995). Generalized Projections: A Powerful Approach to Aggregation. 21st International Conference on Very Large Databases, Zurich, Switzerland, pp. 358 – 369.

16. Gupta, H. (1999). Selection and Maintenance of Views in a Data Warehouse. Ph.D. dissertation, Department of Computer Science, Stanford University.

17. Gupta, H. (1997). Selection of Views to Materialize in a Data Warehouse, Proceedings of the Sixth International Conference on Database Theory, Delphi, Greece, pp. 98-112.

18. Haag, S., Cummings M. and McCubbrey D. (2005). Management Information Systems for the Information Age. McGraw-Hill Irwin.

19. Han, J. & Kamber, M. (2001). Data Mining – Concepts and Techniques. Morgan Kaufmann.

20. Harinarayan, V., Rajaraman, A. & Ullman J.D. (1996). Implementing Datacubes Efficiently, Proceedings of the ACM-SIGMOD International Conference on Management of Data, Montreal, Canada, pp. 205-216.

21. Harinarayan, V., Rajaraman A. & Ullman J.D. (1999). Implementing Datacubes Efficiently, Materialized Views: Techniques, Implementation and Applications, Ed. Gupta, A. and Mumick, I. MIT Press, pp. 343-360.

22. Hass, P. J., Naughton, J. F., Seshadri, S., & Stokes, L. (1995). Sampling-Based Estimation of the Number of Distinct Values of an Attribute. Proceedings of the 21st VLDB Conference, Zurich, Switzerland, pp. 311-322.

23. Haas, P. J. & Stokes, L. (1998). Estimating the Number of Classes in a Finite Population. Journal of the American Statistical Association, 93(444). Theory and Methods, pp.1475-1487.

24. Hillard, R., Blecher, P. and O'Donnell, P. (1999). The Implications of Chaos Theory on the Management of a Data Warehouse. International Society for Decision Support Systems.

25. International Data Corporation (1996). Financial Impact of Data Warehousing.

26. Inmon, W. H. (2002). Building the Data Warehouse. John Wiley & Sons, Inc.

27. Jacobson, R. (2000). Microsoft SQL Server 2000 Analysis Services – Step by Step. Microsoft Press.

28. Kalnis, Panagiotis (2002). Static and Dynamic View Selection in Distributed Data Warehouse System. Ph.D. dissertation, Computer Science, The Hong Kong University of Science and Technology.

29. Kelly S. (1997). Data Warehousing in Action. Wiley, New York.

30. Kimball, Ralph (2002). The Data Warehouse Toolkit. John Wiley & Sons, Inc.

31. Kimball, Ralph (1998). The Data Warehouse Lifecycle Toolkit. John Wiley & Sons, Inc.

32. Kelton, W. D., Sadowski, R. P., Sadowski, D. A. (2002). Simulation with Arena.

33. Medio, A. (1992). Chaotic Dynamics: Theory and Applications to Economics. Cambridge University Press, Cambridge, UK.

34. Mumick, I. Quass, D. and Mumick, B. (1997). Maintenance of Data Cubes and Summary Tables in a Warehouse. Proceedings of ACM SIGMOD International Conference of Management of Data, Tucson, Arizona.

35. Olken, F. (1993). Random Sampling from Databases. Ph.D. dissertation, Department of Computer Science, University of California at Berkeley.

36. O’Neil, P. and Graefe, G. (1995). Multi-table Joins through Bitmapped Join Indices. SIGMOD Record, 24(3), pp.8-11.

37. Qian, X. and Wiederhold, G. (1991). Incremental Recomputation of Active Relational Expressions, IEEE Transaction on Knowledge and Data Engineering, 3(3), 337-341.

38. Ross, K. A., Srivastava, D. & Sudharshan, S. (1996). Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada.

39. Shlosser, A. (1981). On Estimation of the Size of the Dictionary of a Long Text on the Basis of a Sample. Engineering Cybernatics, 19, pp. 97-102.

40. Segev, A. & Fang, W. (1991). Optimal Update Policies for Distributed Materialized Views. Management Science, 37(7), pp. 851-870.

41. Shukla, A., Deshpande, P., & Naughton, J. (1998). Materialized View Selection for Multidimensional Datasets. Proceedings of the 24th Very Large Databases Conference, New York, USA, pp. 488-499.

42. Shukla, A., Deshpande, P., Naughton, J. & Ramasamy, K (1996). Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies. Proceedings of the 22nd Very Large Databases Conference, Mumbai, India, pp. 522-531.

43. Walton, H., Goodhue, D. L., & Wixom, B. H. (2002). The Benefits of Data Warehousing: Why Some Organizations Realize Exceptional Payoffs. Information & Management; 39, pp. 491-502.

44. Watson, H. J. and Haley, B. J. (1997). Data Warehousing: A Framework for Analysis and a Survey of Current Practices. Journal of Data Warehousing, 2(1), pp.10-17.

45. Whalen, E., Garcia, M., DeLuca, S., and Thompson, D. (2001). Microsoft SQL Server 2000 – Performance Tuning. Microsoft Press.

46. Yan, W., & Larson, P. (1995). Eager Aggregation and Lazy Aggregation. Proceedings of the Twenty-First International Conference on Very Large Databases, pp.345-357.