

# OLAP Query Processing for Partitioned Data Warehouses

Ladjel Bellatreche  
*Department of Computer Science*  
*University of Science and Technology*  
*Clear Water Bay, Kowloon*  
*Hong Kong, CHINA*  
*E-mail: ladjel@cs.ust.hk*

Kamalakar Karlapalem  
*Department of Computer Science*  
*University of Science and Technology*  
*Clear Water Bay, Kowloon*  
*Hong Kong, CHINA*  
*E-mail: kamal@cs.ust.hk*

Mukesh Mohania  
*Department of Computer Science*  
*Western Michigan University*  
*Kalamazoo, MI 49008-5021, U.S.A.*  
*E-mail: mohania@cs.wmich.edu*

## Abstract

*On-line analytical processing (OLAP) queries can take hours or even days to execute on very large data warehouses. Therefore, there is a need to employ techniques that can facilitate efficient execution of these queries. Data partitioning concept that has been studied in the context of relational databases aims to reduce query execution time and facilitate the parallel execution of queries. In this paper, we develop a framework for applying the partitioning technique on DW schema (star schema) to reduce the total query execution cost. We develop an analytical cost model for executing a set of OLAP queries on a partitioned star schema. We conduct experiments to evaluate the utility of partitioning in efficiently executing OLAP queries. Finally, we show how partitioning can be used to facilitate parallel execution of OLAP queries.*

## 1 Introduction

Data warehousing (DW) and Online Analytical Processing (OLAP) are becoming critical components of decision support. DWs are very large, and the process of analysis usually involves queries that need aggregates, filters, and grouping of the data in a variety of ways [12]. Querying the DW is very complex, and processing these queries may take hours and days. Note that the analyst or the manager using the DW has time constraints, i.e., requires the answer to his/her queries on time, otherwise he/she cannot make a decision for his/her company. A lot of work has been done to

speed up the OLAP query processing in DWs. Some of the techniques employed are: materialized view [9], advanced indexes [12], sampling and parallel computing technologies [7]. Note that each technique cited above represents a research area by itself. To get a good performance from a DW system, we need a combination of one or more of these techniques.

In this paper, we will concentrate on the data partitioning aspect that can be used for efficiently executing OLAP queries [11]. We will show how we can partition a DW, in order to parallelize the processing of OLAP queries.

DWs require a new-centric view of the data; star schemas which are recommended by Informix [6] provides such a view. The basic premise of star schemas is that information can be classified into two groups: facts and dimensions. Fact tables are the core data elements being analyzed and are *very huge*. Dimensions are attributes about the facts and are much smaller than the fact tables.

Consider a star schema consisting of the SALES table representing the Fact table, and CUSTOMER, PRODUCT and TIME representing the dimension tables as shown in Figure 1. Each dimension table has a primary key. The fact table tuples are associated (through foreign-key reference) to each of the three dimension tables.

Data partitioning first applied on the relational databases in the 80's and is a technique aimed of reducing the number of disk accesses for query execution by minimizing accesses to irrelevant data [4, 13]. We distinguish two types of partitioning vertical and horizontal. Vertical partitioning of a relation  $R$  produces vertical fragments, each of the fragments contains a subset of  $R$ 's attributes as well the primary key

of  $R$ . Horizontal partitioning (HP) partitions a relation  $R$  along its tuples. Each horizontal fragment (HF) has a subset of the tuples of the relation  $R$ . In this paper, we concentrate on HP and discard the vertical partitioning because it is not allowed by the implementation rules of TPC-D benchmark [14]. We will show how we can incorporate the HP in the DW in order to speed up the query processing.

## 1.1 Motivation

TPC-D benchmark [14] in its implementation rules allowed the utilization of HP which has been around for more than ten years, but there has been very little work done in *quantifying* the benefit of using it for speeding up a set of OLAP queries. On the other hand, the vertical partitioning concept which is not allowed by the TPC benchmark D [14] was applied by the DW community in building indexes e.g., projection index [12]. Note that typical OLAP queries access the fact table and dimension tables, and these queries are very expensive, because the size of the fact table is very huge (as such as 10, 000 GB [14]). The main reasons for using the HP in DW environments are:

1. Even though indexing can help in providing good access support at the physical level, the number of irrelevant data retrieved during the query processing can still be very high. The HP aims to reduce irrelevant data accesses [2, 13].
2. Since the OLAP queries use joins of multiple dimension tables and a fact table, the derived HP developed in the relational databases can be used to efficiently process joins across multiple relations [4].
3. Parallelism is a good technique to speed up the OLAP query execution. With a partitioned star schema, we can associate each star schema to one machine and execute queries in parallel.

## 1.2 Contributions and Organization of the Paper

To the best of our knowledge, this is the first paper incorporating the concept of HP in DW applications to speed up certain OLAP queries and to facilitate the parallelism. The main contributions of this paper are that we:

- decompose a star schema into a set of star schemas using the derived HP.
- develop a cost model for executing the most frequent OLAP queries on partitioned star schemas.
- develop a strategy for executing queries on a partitioned star schemas.

- show the utility of HP through evaluation of a given set of OLAP queries.
- present an architecture for facilitating parallel execution of OLAP queries.

The rest of the paper is organized as follows. In Section 2, we present an overview of horizontal partitioning concept. In Section 3, we show the applicability of HP in DWs. A cost model for executing OLAP queries in partitioned DWs is presented in Section 4. Experiments evaluation of partitioning technique and some issues related to the performance of queries are presented in Section 5. Finally, we present our conclusion and some open issues in Section 6.

## 2 HP in Relational Databases

HP consists of partitioning the tuples of a global relation into subsets called horizontal fragments (HFs), where each HF can contain data which have common properties [4]. Two versions of HP are cited by the researchers [4, 13]: primary HP and derived HP. Primary HP of a relation is performed using predicates that are defined on that relation. On the other hand, derived HP is the partitioning of a relation that results from predicates defined on another relation.

Formally, the primary HP is defined as follows: given a relation  $R(K, A_1, A_2, \dots, A_n)$  with each attribute  $A_i$  ( $1 \leq i \leq n$ ) has a domain of values  $Dom(A_i)$ . HF is a subset of the tuples of the relation  $R$ . The subset of tuples forming the HF *must satisfy* a predicate clause.

The derived HP is defined as follows: given two relations  $R$  and  $S$ , with  $S$  containing foreign key of  $R$ , let  $R$  be horizontally partitioned into set of HFs  $\{R_1, R_2, \dots, R_m\}$ , where each HF  $R_i$  ( $1 \leq i \leq m$ ) is given by:  $R_i = \sigma_{cl_j}(R)$ , where  $cl_j$  represents a predicate clause. Then  $S$  can be derived horizontally partitioned into HFs  $\{S_1, S_2, \dots, S_m\}$ , where each  $S_i$  is given by:  $S_i = S \text{ S J N } R_i$ , where ( $1 \leq i \leq m$ ), and **SJN** is the semi join operation [4]. That is, each  $S_i$  contains all those tuples of  $S$  which join with  $R_i$ . The main advantage of derived HP is that it reduces the processing cost of joining relations  $R$  and  $S$ . The reconstruction of a global relation  $R$  from its HFs  $\{R_1, R_2, \dots, R_m\}$  is performed by the union operation among HFs. Thus,  $R = \cup_{i=1}^m R_i$ .

There are a number of techniques for HP, and the choice of how to partition the data can have a real impact on the performance of several types of queries. In this paper, we will not propose a new algorithm for HP, but we will discuss how the HP can be applied in the DW environment.

The main drawback of utilization of the HP in the distributed relational or object oriented databases is the handling the update that may cause the reorganization of the HP schema. But since typical data warehouses are only periodically updated in a batch fashion [12], during which

time the data warehouses are unavailable for querying. This situation makes the utilization of HP feasible, as the reorganization of the HP schema can be done off-line.

## 2.1 Motivating Example

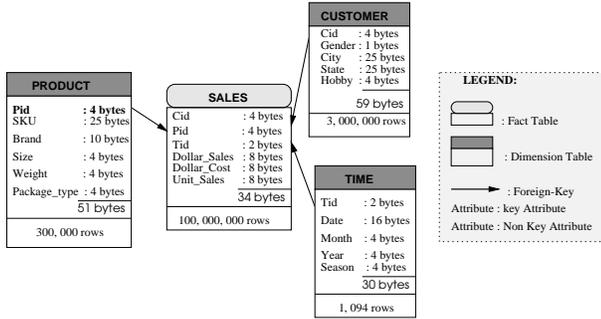


Figure 1. An Example of a Star Schema

We start with an example based on the star schema from [6] to motivate HP. The schema consists of three dimension tables *CUSTOMER*, *PRODUCT*, and *TIME*, and one fact table *SALES*. The tables and attributes of the schema are shown in Figure 1. Let us assume that *only* the dimension table *CUSTOMER* is horizontally partitioned into two HFs *Customer<sub>1</sub>* & *Customer<sub>2</sub>* such that:

*Customer<sub>1</sub>* :  $\sigma_{Gender='M'}(CUSTOMER)$  and  
*Customer<sub>2</sub>* :  $\sigma_{Gender='F'}(CUSTOMER)$ . Based on these HFs, we can derived horizontally partition the fact table *SALES* into two derived HFs *Sales<sub>1</sub>* and *Sales<sub>2</sub>* such that:

*Sales<sub>1</sub>* : *SALES* SJN<sub>Cid=Cid</sub> *Customer<sub>1</sub>* and  
*Sales<sub>2</sub>* : *SALES* SJN<sub>Cid=Cid</sub> *Customer<sub>2</sub>*.

After partitioning *SALES* and *CUSTOMER* tables, our star schema can be represented by two star schemas *S<sub>1</sub>* : (*Sales<sub>1</sub>*, *Customer<sub>1</sub>*, *PRODUCT*, *TIME*) and *S<sub>2</sub>* : (*Sales<sub>2</sub>*, *Customer<sub>2</sub>*, *PRODUCT*, *TIME*), where *S<sub>1</sub>* and *S<sub>2</sub>* represent all sales activities for only the males customers and female customers, respectively. Suppose we have five frequently asked OLAP queries taken from Informix paper [6] on the data warehouse listed in Figure 2. Note that each query can be represented by its query graph [3]. Therefore, we will have five query graphs representing the queries which can be merged to one query graph called *multiple view processing plan* (MVPP) [16] (see Figure 3)<sup>1</sup>. Note that MVPP has multiple root nodes (each corresponding to a query result) and as leaf nodes the fact and dimension tables. The intermediate nodes denote the relational algebra operations performed on the operands. MVPP is used to visualize the commonality of data access among the

<sup>1</sup>The query access frequencies are labeled on the top of each query node in Figure 3.

- $Q_1$ : Select sum(S.dollar\_sales)  
From CUSTOMER C, SALES S  
And C.State = "IL" Selectivity factor =  $\frac{1}{50}$   
And C.Cid = S.Cid  
Group by Cid
- $Q_2$ : Select sum(S.dollar\_sales), sum(S.Unit\_sales)  
From SALES S, PRODUCT P, TIME T  
Where S.Pid = P.PiD  
And S.Tid = T.Tid  
And P.Package\_type = "Box" Selectivity factor =  $\frac{1}{25}$   
And T.Month = "March" Selectivity factor =  $\frac{1}{36}$   
Group by PiD
- $Q_3$ : Select sum(S.dollar\_sales), sum(S.Unit\_sales)  
From SALES S, PRODUCT P, CUSTOMER C, TIME T  
Where S.Cid = C.CiD  
And S.Pid = P.PiD  
And S.Tid = T.Tid  
And P.Package\_type = "Box" Selectivity factor =  $\frac{1}{25}$   
And T.Month = "March" Selectivity factor =  $\frac{1}{36}$   
And C.Gender = 'M' Selectivity factor =  $\frac{1}{2}$   
Group by PiD
- $Q_4$ : Select sum(S.dollar\_sales)  
From SALES S, TIME T, CUSTOMER C  
Where S.Tid = T.Tid  
And S.Cid = C.Cid  
And C.State = "IL" Selectivity factor =  $\frac{1}{50}$   
And T.Month = "March" Selectivity factor =  $\frac{1}{2}$   
Group by PiD
- $Q_5$ : Select sum(S.dollar\_sales)  
From SALES S, PRODUCT P, CUSTOMER C  
Where P.Package\_type = "Box" Selectivity factor =  $\frac{1}{25}$   
And C.Gender = 'M' Selectivity factor =  $\frac{1}{2}$   
And S.Pid = P.PiD  
And S.Cid = C.Cid  
Group by PiD

Figure 2. Example Queries for Data Warehousing Application

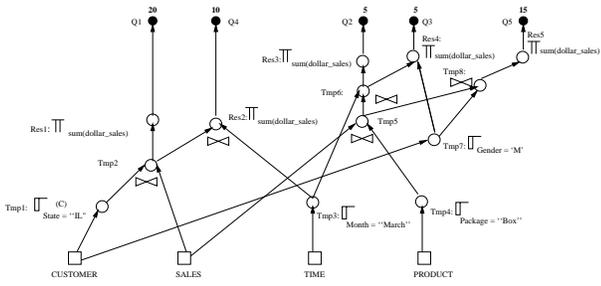


Figure 3. An MVPP for Unpartitioned DW

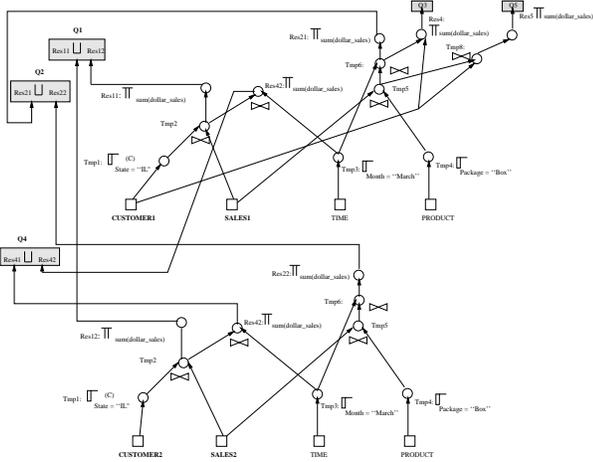


Figure 4. MVPP for Partitioned DW

different queries and helps develop query execution plans for efficiently executing the given set of OLAP queries. After the partitioning process, we will have a partitioned MVPP (PMVPP) as shown in Figure 4. Based on PMVPP, we note that:

- The queries  $Q_3$  and  $Q_5$  access only  $Customer_1$  (Gender = 'M') and therefore  $Sales_1$ . In this case, both of these queries access only the star schema  $S_1$ .
- The queries  $Q_1$ ,  $Q_2$  and  $Q_4$  access the whole schema  $S = S_1 \cup S_2$ . In this case, we can execute these queries in parallel.

### 3 Partitioned Star Schema

We assume we have  $d$  dimension tables  $\{D_1, D_2, \dots, D_d\}$  and one fact table  $F$ . Suppose that each dimension table  $D_i$  ( $1 \leq i \leq d$ ) is horizontally partitioned into  $m_i$  HFs  $\{D_{i1}, D_{i2}, \dots, D_{im_i}\}$ . In this case, the fact table can be derived horizontally partitioned into  $N$  ( $N = \prod_{i=1}^d m_i$ ) HFs [2]. Using this approach of partitioning the number of HFs of the fact table can get

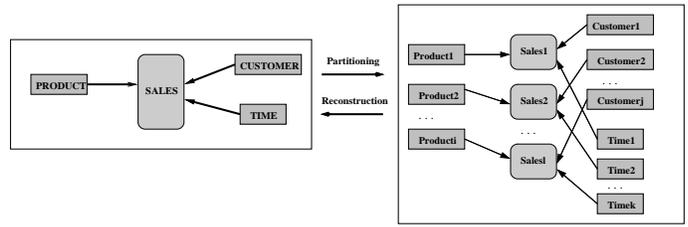


Figure 5. From Unpartitioned Star Schema to Partitioned Star Schema

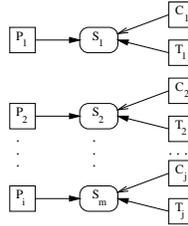


Figure 6. Simple join graph

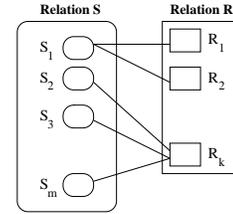


Figure 7. Partitioned join graph

very large. But by using the query-driven approach for partitioning, we can use the frequently asked queries and their access patterns to reduce the number of HFs of the fact table. To do so, we use the merging technique described in [2]; i.e., if  $k$  ( $k \geq 2$ ) HFs of the fact table are accessed by the same queries, then we can merge these  $k$  HFs to one HF.

### 3.1 Affect of Partitioning on Data Warehouses

Before describing the benefit of partitioning on DWs, some concepts related to the partitioning in relational databases are presented.

Assuming that a relation  $S$  is derived horizontally partitioned based on the HFs of a relation  $R$ . A join between the HFs of  $R$  and  $S$  is called a distributed join [4] which can be represented efficiently using join graphs [4]. The join graph of the distributed join between  $R$  and  $S$  is a graph  $G = (V, E)$ , where the nodes  $V$  represent the HFs of  $R$  and  $S$  and an edge between a HF  $R_i$  of  $R$  and  $S_j$  of  $S$  exists if these two HFs are joined together.

When every HF of the relation  $S$  joined to *only one* HF of the relation  $R$ , the join graph is called a *simple join graph* (see Figure 6).

In the DW context, when the fact table is derived horizontally partitioned based on the dimension tables, we will

never have a partitioned join (i.e., the case wherein a HF of the fact table can be joined with more than one HF of the dimension table as shown in Figure 7 will not occur) and we will have only simple join as given by the following theorem.

**Theorem 1** *Let  $F$  and  $D_i$  be a fact table and a dimension table of a given star schema, respectively. The dimension table  $D_i$  is horizontally partitioned into set of disjoint HFs let say,  $\{D_{i1}, D_{i2}, \dots, D_{im_i}\}$ , where each HF is defined by clause predicate:  $D_{ij} = \sigma_{cl_j}(D_i)$ . The fact table  $F$  is derived partitioned based on the HFs of  $D_i$ . Then, the distributed join between  $F$  and  $D_i$  is always represented only by simple join graph.*

**Proof 1** *We prove it by contradiction. Assume that there exists a HF  $F_p$  of the fact table that can be joined to two HFs  $D_{ij}$  and  $D_{il}$  ( $l \neq j$ ) of the dimension table  $D_i$ . Under this assumption we have:*

$$F_p \bowtie D_{ij} \neq \emptyset \quad (1)$$

$$F_p \bowtie D_{il} \neq \emptyset \quad (2)$$

*Equations (1) and (2) are contradictory to the definition of the HF  $F_p$  which is defined as:  $F_p \text{ S J N } D_i$ . Note that HFs of  $D_i$  are disjoint, and  $F_p$  is generated by using the semi-join of foreign key of  $F$  with primary key of  $D_i$ . Therefore, only one of  $F_p \bowtie D_{ij} \neq \emptyset$  or  $F_p \bowtie D_{il} \neq \emptyset$  holds for HF  $F_p$ . Since  $F_p$  can be any HF of  $F$ , each HF of  $F$  joins with exactly one HF of  $D_i$ . This gives rise to simple distributed join graph. The same is true when more than one dimension table is horizontally partitioned and  $F$  is derived horizontally partitioned based on the all the HFs of the dimension tables.*

In star schema modeled data warehouse environments, any fact table  $F$  that is derived horizontally partitioned based on HP schema of dimension tables  $\{D_1, D_2, \dots, D_d\}$  will result in simple distributed join graph. This has two advantages:

- It avoids costly total distributed join (i.e., every HF  $F_p$  ( $1 \leq p \leq N$ ) of the fact table  $F$  joins with each and every HF  $D_{ik}$  of each and every dimension table  $D_i$  ( $1 \leq i \leq d$  and  $1 \leq k \leq m_i$ )).
- It facilitates parallel processing of multiple simple distributed joins (each HF  $F_i$  of the fact table joins with exactly one HF  $D_{ik}$  of  $D_i$ ).

Note that in general derived HP can give rise overlapping HFs [4] and hence a total distributed join, this costly distributed join is avoided in case of partitioned DW.

### 3.2 Query Execution Strategy in Partitioned Star Schema

Since the dimension tables and fact table are horizontally partitioned, we need to ensure the data access transparency concept in which a user of the DW is purposefully unaware of the distribution of the data. Our goal is to provide to the DW users the unpartitioned star schema and the query optimizer task is to translate the OLAP queries on the unpartitioned star schema to partitioned star schemas. In this section, we will show how the query gets executed in a partitioned star schema.

**Definition 1 (Relevant Predicate Attribute)** *is an attribute which participates in a predicate which defines a HP. Any attribute which does not participate in defining a a predicate which defines a HP is called irrelevant predicate attribute.*

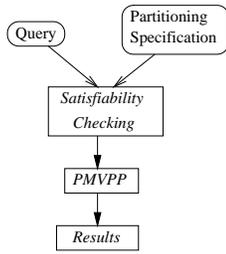
Suppose we have  $N$  partitioned star schemas  $\{S_1, S_2, \dots, S_N\}$ , where each dimension table  $D_{jk}$  and fact table  $F_i$  of  $S_i$  is a HF of the dimension table  $D_j$  and the fact table  $F$ , respectively. Each dimension table  $D_{ij}$  has its partitioning condition which is a predicate clause. For example the dimension table  $Customer_1$  of the star schema  $S_1$  (see Section 2.1) is represented by the following partitioning condition : Gender = 'M'. The same thing for the fact table  $F_i$  which is defined by the semi join condition as shown in Table 1. We call these partitioning conditions *the partitioning specification* which can be implemented using a table.

From the above definition, we can say that all relevant predicate attributes of a given partitioned star schema are the attributes in the partitioning specification. We assume

HF	HP Condition
$Customer_1$	Gender = 'M'
$Customer_2$	Gender = 'F'
$Sales_1$	SALES S J N $Customer_1$
$Sales_2$	SALES S J N $Customer_2$

**Table 1. Partitioning Specification Table**

we have an OLAP query  $Q$  with  $P_\sigma$  selection predicates and  $P_\bowtie$  operations executed on a partitioned star schema  $S = \{S_1, S_2, \dots, S_m\}$ . For each selection predicate  $SP_i$ , we define the function  $attr(SP_i)$  which give us the name of the attribute used by  $SP_i$ . The union of the  $attr(SP_i)$  gives us the names of all attributes used by  $Q$ . We call this set query predicate attributes(QPA). Using the partitioning specification table with its relevant predicate attribute (RPA) and QPA, we can distinguish three major scenarios:



**Figure 8. Identifying the HFs Needed for an OLAP Query**

1. If  $(QPA = \emptyset)$  meaning that the query  $Q$  does not contain selection predicates. We have two approaches to execute the query  $Q$ :
  - (a) Perform the union operations of all star schemas and then perform the join operations as in unpartitioned star schema.
  - (b) Perform the join operations in all star schema and then assemble the result using the union operation.
2. if  $(QPA \neq \emptyset)$  and  $(RPA \cap QPA = \emptyset)$  which means that query  $Q$  has some selection predicates on only unpartitioned dimension table(s). To execute the join we use the above choices (1.a) and (1.b).
3. if  $(RPA \cap QPA \neq \emptyset)$  means that some query attributes in selection predicates match with RPA. In this case, we can easily determine the names of the corresponding dimension table(s) which were HP. We can also determine the relevant HFs for the query  $Q$ ; i.e., verify if the query selection predicates matches with the HP predicates. This problem is known as *satisfiability problem* [8] as shown in Figure 8. Guo et al. [8] develop an efficient algorithm to solve the satisfiability problem. We use their approach to address this problem.
4. if  $(QPA \subseteq RPA)$  meaning that all query attributes match with the RPA. In this case, the query  $Q$  may get executed very fast. We call this case the “best case”.

## 4 Query Processing Cost

In this section, we present two cost models for processing an OLAP query, the first one is for unpartitioned star schema, and the second one is for partitioned star schema. These cost models are used for executing a set of frequently asked queries in the DW  $\{Q_1, Q_2, \dots, Q_n\}$ . The objective of our cost models is to calculate the cost of executing these

queries in terms of disk page accesses (IO cost) during the selection and join operations (which are the most used and most expensive operations in data warehouses [10]).

First, we develop a cost model for unpartitioned star schema (all tables are not partitioned). Let  $Q_j$  be an OLAP query having  $\alpha$  selection predicates and  $\beta$  join operations. Suppose that the order of join of an OLAP query is given by MVPP [16]. To perform the join, we use the hash join technique.

The cost of executing a query  $Q_j$  is the sum of the cost of selection operations ( $SC$ ) and cost join operations ( $JC$ ). To perform the selection and join operations, we make two major assumptions regarding the size of the main memory, as it affects the number of disk accesses required [15]:

1. *Large Memory Hypothesis (LMH)*: All dimension tables are in the main memory because their sizes are very small [5], and the fact table is the disk, but the loading of the fact table from the disk to the main memory is done only once. This assumption becomes more and more realistic as the size of main memory keeps increasing because of fall in main memory prices.
2. *Medium Memory Hypothesis (MMH)*: It is similar to the LMH, but the single difference is that sometimes the size of the intermediate result cannot fit in the memory, and then we need to store it in the disk and reload it if required.

For lack of space, we cannot describe our cost models. For more details (see [1]).

## 5 Evaluation of Partitioned Data Warehouses

In order to show the utility of HP in the DW context given by a data warehouse schema and data set from Informix [6], we conduct some experiments study to show the tradeoff of partitioned and unpartitioned DWs. As in section 2, the schema consists of three dimension tables *CUSTOMER*, *TIME*, and *PRODUCT* and one fact table *SALES*. The key characteristics of our experimental data warehouse are as shown in Table 2. The selectivity factors corresponding to the five frequently asked queries are shown in Figure 2.

To characterize the improvement of performance using the HP technique, we defined a normalized IO metric as follow:

$$\text{Normalized IO} = \frac{\# \text{ Of IOs for the Horizontally Partitioned Star Schema}}{\# \text{ Of IOs for the Unpartitioned Star Schema}}$$

We note that if the value of normalized IO is less than 1.0, then it implies that HP is beneficial. Let assume that the possible HP schemas for all three dimension table are:

Parameter	Description	Value
<i>SALES</i>	Number of rows	10000000
w( <i>SALES</i> )	<i>SALES</i> width (in bytes)	34
<i>CUSTOMER</i>	Number of rows	3000000
w( <i>CUSTOMER</i> )	<i>CUSTOMER</i> width (in bytes)	59
<i>PRODUCT</i>	Number of rows	300000
w( <i>PRODUCT</i> )	<i>PRODUCT</i> width (in bytes)	51
<i>TIME</i>	Number of rows	1094
w( <i>TIME</i> )	<i>TIME</i> width (in bytes)	30
<i>PS</i>	Page Size (in bytes)	8192

**Table 2. Parameters used in the Experiments**

- *CUSTOMER* is partitioned using Gender attribute into two HFs  $Customer_1$  : Gender = ‘M’ and  $Customer_2$  : Gender = ‘F’.
- *TIME* is partitioned using Season attribute into four HFs  $Time_1$  : Season = “Winter” and Season = “Summer”.
- Product is partitioned using Package\_type attribute into two HFs  $Product_1$  : Package\_type = “Box” and  $Product_2$  : Package\_type = “Paper”

We assume that the size of HFs issued from the dimension tables and the fact table are uniformly distributed.

To see the impact of the number of dimension tables which are horizontally partitioned, we will concentrate on the following three cases:

1. **Case 1:** Only the dimension table *CUSTOMER* is partitioned and then *SALES* is derived horizontally partitioned into two HFs  $Sales_1$  and  $Sales_2$ .
2. **Case 2:** Both dimension tables *CUSTOMER* and *PRODUCT* are horizontally partitioned; then *SALES* can be partitioned into four HFs.
3. **Case 3:** All dimension tables are partitioned, in this case *SALES* can be partitioned 8 HFs.

Based on the HP schemas defined above, as well as the five frequently asked queries, we evaluate the HP under LMH and MMH. Let  $(TC_i)$  and  $(NIO_i)$  be the total cost in terms of IOs for each given query and the normalized IO for in each given query, respectively.

### 5.1 Partitioned Star Schema under LMH

Note that under LMH, we load the fact table only once, and then we evaluate all operations(selections and joins). Based on Table 3, we get the following observations:

- The HP gives better result in comparison to the unpartitioned case. In the worse case, it gives the same result as for the query  $Q_1$ . This is because  $Q_1$  has a QPA (C.State = “IL”) that is different from the RPA of the dimension table *CUSTOMER* which is based on the attribute Gender.
- The best case of partitioning is when the following condition is satisfied ( $QPA \subseteq RPA$  (see Section 3)). Query  $Q_3$  is an example of such query and we notice in Table 3 that it gets executed efficiently in all cases.
- The number of dimension tables that are horizontally partitioned plays an important role in the performance. As we see in Table 3, when all dimension tables are horizontally partitioned(case 3), almost all queries benefit from the HP technique. On the other hand, when we have only one dimension table horizontally partitioned, we cannot see the benefit of HP(case 1). The disadvantage is that when all dimensions tables are partitioned; the number of HFs of the fact table may be very large.

Q	TC	TC <sub>1</sub>	TC <sub>2</sub>	TC <sub>3</sub>	NIO <sub>1</sub>	NIO <sub>2</sub>	NIO <sub>3</sub>
Q1	1245120	1245120	1245120	1245120	1.00	1.00	1.00
Q2	1245120	1245120	622560	311280	1.00	0.5	0.25
Q3	1245120	622560	311280	155640	0.25	0.25	0.125
Q4	1245120	1245120	1245120	622560	1.00	1.00	0.5
Q5	1245120	622560	311280	311280	0.5	0.25	0.25

**Table 3. Query Processing Cost Unpartitioned & Partitioned Star Schema Under LMH**

### 5.2 Partitioned Star Schema under MMH

In this section, we will see the effect of MMH on the performance. Note that, under the MMH, the union operation plays an important role in decreasing the performance and sometimes the HP perform badly. For example, the cost of executing the query  $Q_1$  on unpartitioned star schema is **1245120** and on partitioned star schema is **2075200** for case 1 and the normalized IO is **1.67** as shown in Table 4. This is due to the union operation. But generally, in almost all cases the HP performs better than the unpartitioned case.

For the queries which have a bad performance under a partitioned star schema and to avoid the intensively cost of union operation, we can execute these queries in parallel.

## 6 Conclusion and Future Work

As data warehouses(DWs) become larger and larger, the amount of time taken to execute complex OLAP queries

Q	TC	TC <sub>1</sub>	TC <sub>2</sub>	TC <sub>3</sub>	NIO <sub>1</sub>	NIO <sub>2</sub>	NIO <sub>3</sub>
Q1	1245120	2075200	1660160	1037600	1.67	1.34	0.83
Q2	3320320	3320320	1867680	933840	1.0	0.56	0.28
Q3	5395520	2282720	1141360	570680	0.42	0.21	0.1
Q4	3320320	3320320	1867680	1867680	1.0	0.56	0.56
Q5	3320320	1452640	726320	933840	0.43	0.22	0.28

**Table 4. Query Processing Cost Unpartitioned & Partitioned Star Schema Under MMH**

gets larger. Hence, there is a need for developing techniques that can facilitate efficient OLAP query execution for large DWs. In this regard, we take a fresh look at data partitioning and show its utility in efficiently executing OLAP queries. In particular, we find that DWs to be amenable to horizontal partitioning and that star schema facilitates simpler and parallelizable distributed join graphs. We consider a large DW schema and parameters from Informix white paper, and show that horizontally partitioned data warehouse does facilitate efficient execution of OLAP queries. Moreover, we show that it is better to horizontally partition all the dimension tables, rather than only one. Finally, we show that medium memory size hypothesis can make it costly to execute queries in partitioned DW in comparison to unpartitioned data warehouse. Whereas, under large memory hypothesis, the queries at worst get executed with a cost for unpartitioned data warehouse. Thus, in this paper, we have rekindled applicability of horizontal partitioning in data warehousing environments and showed its utility in executing OLAP queries. Further, we have introduced two issues regarding parallelizing OLAP queries on partitioned DW, and reorganization of partitioned DW based on the changes in query access patterns and database updates.

## References

- [1] L. Bellatreche and K. Karlapalem. Olap query processing for partitioned data warehouses. Technical Report HKUST-CS99-08, May 1999.
- [2] L. Bellatreche, K. Karlapalem, and Simonet A. Algorithms and support for horizontal class partitioning in object-oriented databases. *Accepted for publication in Distributed and Parallel Databases: an International Journal, Kluwer Academic Publishers*, September 1998.
- [3] P. A. Bernstein and D-M. W. Chiu. Using semi-joins to solve relational queries. *Journal of the ACM*, 28(1):25–40, January 1981.
- [4] S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data. SIGPLAN Notices*, pages 128–136, 1982.
- [5] Oracle Corp. Star queries in oracle8. *White Paper*, June 1997.
- [6] Informix Corporation. Informix-online extended parallel server and informix-universal server: A new generation of decision-support indexing for enterprise data warehouses. *White Paper*, 1997.
- [7] A. Datta, B. Moon, and H. Thomas. A case for parallelism in data warehousing and olap. *in the 9th International Workshop on Database and Expert Systems Applications (DEXA98)*, pages 226–231, August 1998.
- [8] S. Guo, S. Wei, and M. A. Weiss. On satisfiability, equivalence, and implication problems involving conjunctive queries in database systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):604–612, August 1996.
- [9] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *Data Engineering Bulletin*, 18(2):3–18, June 1995.
- [10] H. Lei and K. A. Ross. Faster joins, self-joins and multi-way joins using join indices. *Data and Knowledge Engineering*, 28(3):277–298, November 1998.
- [11] D. Munneke, K. Wahlstrom, and Mohania M. K. Fragmentation of multidimensional databases. *in the 8th Australian Database Conference (ADC’99)*, pages 153–164, 1999.
- [12] P. O’Neil and D. Quass. Improved query performance with variant indexes. *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 38–49, May 1997.
- [13] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.
- [14] TPC Home Page. Tpc benchmark<sup>TM</sup> d (decision support). <http://www.tpc.org>.
- [15] R. Ramakrishnan. *Database Management Systems*. WCB/McGraw Hill, 1998.
- [16] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. *Proceedings of International Conference on Very Large Databases*, pages 136–145, August 1997.