

Efficient Query Processing with Associated Horizontal Class Partitioning in an Object Relational Data Warehousing Environment

Vivekanand Gopalkrishnan
Department of Computer Science,
City University of Hong Kong,
Kowloon, Hong Kong, PRC
vivek@cs.cityu.edu.hk

Qing Li
Department of Computer Science,
City University of Hong Kong,
Kowloon, Hong Kong, PRC
csqli@cityu.edu.hk

Kamalakar Karlapalem
Department of Computer Science,
University of Science and Technology
Clear Water Bay, Hong Kong, PRC
kamal@cs.ust.hk

Abstract

In an Object Relational Data Warehousing (ORDW) environment, the semantics of data and queries can be explicitly captured, represented, and utilized based on is-a and class composition hierarchies, thereby resulting in more efficient OLAP query processing. In this paper, we show the efficacy in building semantic-rich hybrid class partitions by incorporating the Associated Horizontal Class Partitioning (AHCP) technique on the ORDW schema. Given a set of queries, we use primary and derived partitioning algorithms to select (near) optimal AHCPs, thereby embedding query semantics into the partitioned framework. Finally, by a cost model, we analyze the effectiveness of our approach vis-a-vis the unpartitioned approach.

1 Introduction

Data warehouse (DW) equips users with more effective decision support tools by integrating enterprise-wide corporate data into a single repository from which business end-users can run reports and perform ad hoc data analysis [CD97]. As DWs contain enormous amount of data, often from different sources, we need highly efficient indexing structures [Sar97], [GHRU97], [VLK00], materialized (stored) Views [Rou97], and query processing techniques [VLK99] to efficiently answer on-line analytical processing (OLAP) queries. Materialized Views represent integrated data based on complex aggregate queries, and should be available

consistently and instantaneously. Maintaining the integrity of these Indexes and Views [GM95], [MK99] imposes a challenging problem when the source data changes frequently, when the size of the DW keeps growing, and/or when the user queries become increasingly complex. An extensible framework that can accommodate dynamic warehousing [Dayal99] of changing data gracefully, and have adaptive handles for processing OLAP queries efficiently is needed.

In [VLK98], we showed that besides establishing a semantically richer framework for multi-dimension hierarchies, the Object Relational View (ORV) model provides excellent support for complex object retrieval. In [VLK99] we presented the Object Relational Data Warehousing (ORDW) approach to address some of the issues discussed in [VLK98] on data warehousing. More specifically, we devised a translation mechanism from the star/snowflake schema to an object oriented (O-O) representation. In [VLK00], we advocated a query processing strategy implementing the Structural Join Index Hierarchy (SJIH) on ORDW.

In this paper, we show the efficacy in building semantic-rich hybrid class partitions by incorporating the Associated Horizontal Class Partitioning (AHCP) technique on the ORDW schema. Given a set of queries, we use primary and derived partitioning algorithms to select (near) optimal AHCPs, thereby embedding query semantics into the partitioned framework. Finally, by a cost model, we analyze the effectiveness of our approach vis-a-vis the unpartitioned approach.

To put our research in perspective, we review some related work and briefly outline our previous work in the field of ORDW and Class Partitioning on OODBs in section 2. We further motivate our study by presenting on the ORDW schema some sample queries whose patterns are classified based on DW operations and by OO concepts. Obtaining an optimal partitioning scheme to process this set of queries is the focus of section 3, where we employ a hill-climbing heuristic algorithm to select a

The copyright of this paper belongs to the paper's authors. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage.

Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2000)

Stockholm, Sweden, June 5-6, 2000

(M. Jeusfeld, H. Shu, M. Staudt, G. Vossen, eds.)

<http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-28/>

(near) optimal AHCPs. This algorithm is profiling driven, and can be further extended to incorporate other semantics. In section 4, we compare results of retrieval costs using the AHCPs vs. the unpartitioned case. Section 5 concludes the paper with a brief look at future work.

2 Background and Motivating Example

2.1 Related Work

Partitioning has been vastly researched in Relational and OO database systems. Excellent work has been done in Vertical Partitioning (VP) and Horizontal Partitioning (HP) in both systems, but the unique features of OO systems have made it possible to experiment with different variations such as Derived Horizontal Class Partitioning (DHCP), Associated Horizontal Class Partitioning (AHCP), Path Partitioning (PP) and Method Induced Partitioning (MIP). [KL00] presents a comprehensive framework for devising partitioning schemes based on different types of methods and their classification. The issue of fragmentation transparency is addressed by considering appropriate method transformation techniques. While those methods were extremely successful in the transactional environment of an OODB, to the best of our knowledge, no work has been done in partitioning of an Object Relational DB or Object Relational Data Warehouse (ORDW).

Recently, we have conducted some preliminary studies on developing an ORDW framework. In [VLK98], we showed that the ORV (Object Relational View) model offers inherent features that are conducive to managing a data warehouse. We listed the various issues that arise during the design of an OR-DWMS (Object Relational Data Warehouse Management Systems). Here, OR means an object-oriented front-end or views to underlying relational data sources. Based on the issues discussed in [VLK98], we put forward a three-phase design approach in [VLK99], which also provided a query-driven translation mechanism from the star/snowflake schema to an object oriented (O-O) representation. Some query processing strategies utilizing Structural Join Index Hierarchy (SJIH) techniques for complex queries on composite objects were addressed in [VLK00].

2.2 Motivating Query Examples

To further motivate our subsequent discussions, let us consider the sample ORDW schema as shown in Figure 1, taken from [VLK99]. This schema is a simple single-star/snowflake schema, for a sales application. As seen in the figure, dimension classes (DCs) are connected by solid arrows (composition hierarchies) to the main fact class (FC). In addition, there are other inter-dimension hierarchies (as obtained by vertical partitioning¹), and

¹ This paper deals only with Associated Horizontal Partitioning techniques, and the above schema has been arrived at using other techniques.

other composition links. Moreover, we also demonstrate is-a hierarchies (denoted by dotted lines) obtained by horizontally partitioning the ORDW schema.

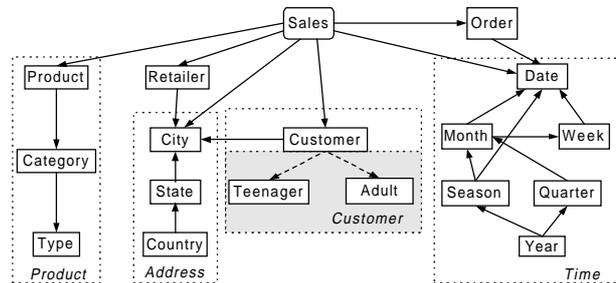


Fig. 1. The ORDW Schema.

The figure shows the class composition hierarchy for the Time dimension, and the is-a hierarchy (shaded area) for the Customer dimension.

A Fact is the “subject” of the OLAP queries, and is quantified by its dimensions and “values”. Dimensions can be hierarchical and composite in nature, whereas Values are numerical data. When a Dimension is complex enough to contain various other components that can themselves be classified as Dimensions and Values then that Dimension can be a “Fact” of another OLAP query. In this case, we can consider the schema to be that of “Nested Fact” or “Fact within Fact”.

Whereas when two (or more) Facts share (one or more) Dimensions, then the OLAP queries can be considered as “Inter-Fact”. To support OLAP applications, we define a group of OLAP queries OQG (modified from [VLK00] by adding predicates), which are invoked as a set (not necessarily in an order). Query patterns in the OQG may not be restricted to a particular composition hierarchy or inheritance hierarchy. The access paths may involve multiple paths emanating from the same complex object, as well as interact with entities in completely unrelated complex objects.

For this discussion, queries involving Nested Facts can be considered as subsets of inter-Fact queries. They are distinguished by the presence of a semantic disjointness between the Facts involved. It must be noted though that this disjointness does not preclude the Facts from sharing the same component objects. A query-processing scheme built on separate Facts will inadvertently need costly joins. This inefficiency is amplified for queries with low selectivity and high frequency. This calls for a need for a partitioning scheme that transcends Facts and is not restricted by the hierarchies mentioned. It must be noted that such a partitioning scheme may well be overlapping and hence will suffer on the storage space.

Based on classifications by DW operations & by OO concepts, we consider the following queries listed in Table 1 as our sample OQG for subsequent discussions.

Table 1. Sample OLAP queries - OQG

No.	Query	Query type
Q1	Sales by Prod by State in US	Only along cch (pivot)
Q2	Sales by Prod by State by Year in US	-> Drill-down
Q3	Sales by Prod for Categ=Elec	-> Roll-up
Q4	Sales by Prod by City for Categ=Elec	Only along cch, Drill-down
Q5	Sales by Prod by Country for Categ=Elec	Only along cch, Roll-up
Q6	Sales by Prod to Teenagers by State for Categ=Elec & in US	Only along cch, Slice_and_dice
Q7	Sales of Prod 1 compared with Sales of Prod 2 to Teenagers for Categ=Elec	Only along cch, Drill-down, Slice_and_dice
Q8	% increase in Sales to Teenagers over Sales to Adults, of Prod 1 / 2 for Categ=Elec & in US	Combination of is-a & cch, Drill-down, Slice_and_dice

Some OLAP queries could be on the entire range of Sales and would need to access multiple dimensions for the "Group BY". However, as seen above, some queries could have a predicate range such as "Categ=Elec" or "Country=US". In such cases, the search space on the Fact "Sales" is reduced by a factor equal to the selectivity of the predicate. However, this does not help during query processing (normal unpartitioned case), as the entire FC is processed while searching for relevant tuples. Even in cases where indexes are built [VLK00], the benefit could be reduced, as index creation takes up more time due to the enormity of the FC. Further, as the OLAP queries involve multiple paths (multiple selections and group bys), the size of the Forward and Reverse Joins is dependent on the size of the Root (FC). This calls for the need to partition the FC according to the query characteristics.

3 AHCP Selection Methodology

The Associated Horizontal Class Partitioning (AHCP) methodology creates semantic-rich hybrid class partitions for efficient query processing. It is a technique by which several classes can be partitioned according to the semantics of another class in its aggregation hierarchy. We employ the AHCP on our ORDW schema, and propose to extend its applicability from class composition hierarchies to also include is-a hierarchies and links quantified by partial participation, thereby encompassing the Complete Warehouse Schema (CWS) in the ORDW.

3.1 AHCP fundamentals

The total cost of the AHCP framework can be broadly categorized as partition storage cost, retrieval cost and maintenance cost. In this paper, we also incorporate query-centric information including selectivity and

frequency to determine the selection of minimal complete set of partition fragments for optimal storage, maintenance and retrieval costs.

3.1.1 Primary Horizontal Partitions (PHP)

Classes in the ORDW schema can be denoted as C_i^p , which indicates the i^{th} class in the p^{th} path. The root class (FC) is denoted as C_0 . Primary Horizontal Partitions on these classes can be denoted as sub-classes and placed in the is-a hierarchy under the original partitioned class. Note that the (sub) is-a hierarchy in our examples is denoted by the subscript $i.j$, which denotes the j^{th} sub-class of the i^{th} class (in the p^{th} path). The Primary Horizontal Partitioning (PHP) operation is denoted as :

$$PHP(C_i^p)_{p1} \rightarrow \{ C_{i.1}^p, C_{i.2}^p, \dots, C_{i.n}^p \}$$

where (C_i^p) is the Class that is Primary Horizontally Partitioned according to a predicate (p1), resulting in n fragments which are treated as classes $\{C_{i.n}^p\}$. Note however, that since FC is the only root in the realm of our OLAP OQGs, any primary partition of the root need not display the path suffix; ie. $(C_{0.1}^0 = C_{0.1})$.

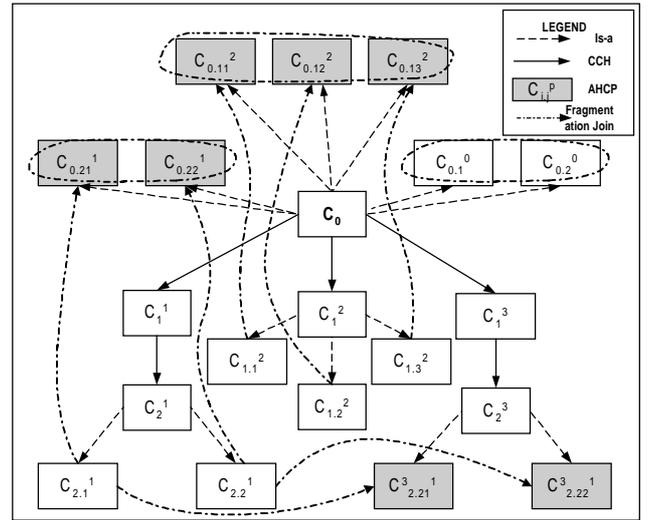


Fig. 2. An AHCP example on the Fact and Dimensions

The example in figure 2 shows classes C_0 , C_2^1 and C_1^2 in the class composition hierarchy (CCH). Some of the PHPs are $\{ C_{1.1}^2, C_{1.2}^2 \text{ and } C_{1.3}^2 \}$, connected by dashed lines (is-a) to the super-class C_1^2 which was partitioned.

As the PHPs can be considered as subclasses of the class on which the PHP were performed, they are placed in the is-a hierarchy of the schema. We can have any no. of PHPs on a single class based on a number of predicates. For a single simple predicate, the PHPs are disjoint, i.e. they don't share any objects. PHP schemes based on multiple or complex predicates on the same class, may induce overlapping fragments, however we do not consider such schemes in this work to avoid complexity.

3.1.2 Associated Horizontal Class Partitions (AHCP)

After the PHPs are created, the AHCP operation may be performed on some other classes in the schema. As noted above, most queries in the OQG access the root (FC) for its value based attributes, and hence this paper deals primarily with AHCP of the root class. The AHCP operation can be denoted as follows :

$$\text{AHCP}(C_j^q, \text{PHP}(C_i^p)_{p1}) \rightarrow \{ C_{j.i1}^q, C_{j.i2}^q, \dots, C_{j.im}^q \}$$

where (C_j^q) is the Class that is Associate Horizontally Class Partitioned (AHCP) according to the PHP on class C_i^p , resulting in m fragments which are treated as classes $\{ C_{j.im}^q \}$. Here again since FC is the only root in the realm of our OLAP OQGs, any AHCP of the root need not display the path suffix; ie. $(C_{0.11}^0 = C_{0.11}^2)$.

As seen in the figure, the examples indicate that two sets of AHCPs are created from the root C_0 . They can be created by :

$$\begin{aligned} \text{AHCP}(C_0, \text{PHP}(C_1^2)_{p1}) &\rightarrow \{ C_{0.11}^2, C_{0.12}^2, C_{0.13}^2 \}; \\ \text{AHCP}(C_0, \text{PHP}(C_2^1)_{p1}) &\rightarrow \{ C_{0.21}^1, C_{0.22}^1 \} \end{aligned}$$

These partition fragments are denoted as subclasses in the figure by means of the shaded boxes to indicate Associate Partitioning.

The AHCP operation can also be performed on classes other than the root, ie. the Dimension Classes. For example, as seen in the figure, C_2^3 can be AHCPed based on the PHPs of C_2^1 .

$$\text{AHCP}(C_2^3, \text{PHP}(C_2^1)_{p1}) \rightarrow \{ C_{3.21}^1, C_{3.22}^1 \}$$

The result is shown in shaded boxes under C_2^3 in the figure.

An important point to be noted here is that while the Fragments obtained by any single AHCP operation on the root are always disjoint, the same cannot be said about Fragments obtained by AHCP on any other (Dimension) class. This indicates the storage overhead to be incurred while performing AHCPs on the Dimension classes, and must be taken into account by the cost model.

3.2 AHCP cost model

In an ORDW, partitioning can be implemented by means of Method Induced Partitioning techniques [KL00]. Moreover, due to the structural and cardinal differences inherent between Dimension Classes (DC) and the Fact Class (FC), we can assume that the DCs need not be physically partitioned as they may be wholly or partially stored in memory (under both medium and large memory hypothesis). Hence, the cost of the traditional *join* between the PHP fragments and the AHCP fragments can be ignored. This *join* can be achieved by employing the methods of the FC.

3.2.1 Storage Cost

The Storage cost (SC) has two components : Primary Horizontal Partition (PHP), and the Associated Horizontal Class Partition (AHCP). It can be stated as :

$$SC = SC_{PHP} + SC_{AHCP}$$

They are given as follows :

1. $SC_{PHP}(C_1)$:

We assume that in most cases, and especially in this paper, we consider only one PHP per class. This ensures that the partitions are disjoint for simple predicates. In such cases, there's negligible overhead for storage cost as $SC_{PHP}(C_1) = |C_1|$ (no. of pages occupied by the class C_1 + catalog entries for the no. of PHPs of C_1). These catalog entries give details of the partitioned Class structure, extent and qualifying rules. Hence, they're very small and can easily be accommodated in memory (in both the medium and large memory hypothesis).

In case of multiple complex predicates on a Dimension (C_1), resulting in overlapping fragments, we propose not to replicate the entire class extent, but rather only replicate the Class OIDs (and some frequently accessed attributes) in the separate Partitions.

In this case, the storage overhead can be estimated as :

$$SC_{PHP}(C_1) = \|C_1\| \times \text{No}_{Attr} \times (\text{sizeof}(Attr)) \times \text{No}_{PHP}$$

where No_{Attr} = No.of Attributes replicated.
where No_{PHP} = No.of Partition schemes.

Given a maximum of 2 replicated attributes or 20% of the class structure, and a uniform size of attributes, we can accommodate up to 5 different Partitioning schemes for an increase of 100% in $SC_{PHP}(C_1)$.

2. $SC_{AHCP}(C_0)$:

This is by far the biggest increment for storage cost in the AHCP ORDW. As noted above, the root (C_0) would be the widely used as the candidate for performing AHCP. Since any predicate on a single dimension can only induce disjoint partitions in the root, the partitioning overhead is negligible for multiple partitioning schemes in a single DC.

$$SC_{AHCP}(C_0) = |C_0| + \text{No}_{PHP} \times \text{Size}_{Cat}(\text{PHP}_i).$$

where Size_{Cat} = Catalog entry size (structure, extent, qualifying rules).

However, as we incorporate multiple predicates on different dimension classes, $SC_{AHCP}(C_0)$ grows linearly as the no. of dimensions (assuming only single complex predicates on each dimension). This can be a large

overhead, as C_0 as the FC, is very large (~order of Gigabytes).

Hence, we intend to reduce this overhead by means of a Multiple Partition Processing Plan (MP^3), based on MVPP [YKL97]. This would entail a compromise between duplication and efficiency of the partitions, as sub-fragments will have to be created to support the AHCPs. The Join needed to produce the final result from these sub-fragments constitutes the increase in retrieval cost.

3.2.2 Maintenance cost

As noted above, since inter-fragment *join* is avoided between the PHP and AHCP fragments, maintenance cost is considerably simplified due to the AHCP operation.

As the ORDW is a read mostly and append only environment, we can safely estimate the maintenance cost even though the schema is vastly enhanced (and complicated) by semantics. For example, once the Warehouse has achieved full functionality, in each update cycle of the ORDW, we can expect up to 0.5% addition of the FC (this is a very conservative estimate based on our same DW, maintaining 10 years worth of "Sales" data and updated daily). The updates to DCs can be ignored mainly because their percentage will be even smaller and also because most of the DCs will be in memory anyway. Only these 0.5% FC objects have to be processed in order to maintain the Partitioning scheme.

The Maintenance cost for the AHCP partitioning scheme (MC) can be defined as the extra cost of maintaining the AHCPs and the PHPs catalogs.

$$MC = MC_{Cat}(AHCP_i) + MC_{Cat}(PHP_i)$$

Since $MC_{Cat}(PHP_i)$ is negligible as the PHPs are in memory, the main cost is on the AHCP maintenance, which is comprised of maintaining catalog entries of the AHCP, Generally this meta-information is small enough to be stored completely in memory.

3.2.3 Retrieval cost

To determine retrieval cost, we break up the complex queries into smaller atomic sub-query expressions. We denote this by means of a MQO (Multiple Query Optimization) graph in the MP^3 , which is further explained in section 3.3.

The Retrieval Cost (RC) is the cost of parsing the catalog, accessing the relevant AHCPs (as union) and the cost of the *join* with corresponding PHPs.

$$RC = RC_{Cat} + RC_{AHCP} + RC_{PHP} + RC_{join}$$

However, as we store the PHPs and the *join* in memory, and the Catalog is relatively small, RC is mainly

composed of AHCP loading cost. Since this is smaller than the complete FC by a factor of $\min(sel_{pi})$, where sel_{pi} indicates the selectivity of the predicates on query Q_i , we achieve a considerable savings in retrieval cost.

This saving is also obtained when indexing schemes like the SJH [VLK00] are built on top of the AHCPs, and also when aggregate views have to be developed.

3.3 AHCP selection procedure

We approach the problem of performing AHCP in the ORDW in a different manner from the case of DHCP in a normal OODB [BKS98]. IN [BKS98], various techniques (candidates) were considered to decide the best PHP candidate for performing DHCP. Here we consider all the PHP candidates, and our AHCP algorithm generates a optimal combination of complete and minimal set of AHCPs.

3.3.1 AHCP Algorithm (also called MP3 algorithm)

The algorithm can be broken into three parts :

1. Generating an exhaustive set of AHCPs based on query characteristics (selectivity, fan-out) obtained from the entire query space.
 - 1.1 For each query Q_i in the OQG , generate logical associated fragments $\{C_{0,j}^p\}$ from $\{C_j^p\}$ that satisfies sub-expressions of Q_i completely.
 - 1.2 Perform an intersection of the $C_{0,j}^p$ fragments for all Q_i . This creates the complete disjoint AHCP set, on which the queries will be based.
2. Assigning query weights depending on priority and importance (frequency).
 - 2.1 For each query Q_i , evaluate the minimal set of query processing fragments : $QPF_i = \{ C_{0,j1}^{p1}, C_{0,j2}^{p2}, \dots, C_{0,jm}^{pm} \}$
 - 2.2 Create query plans for each Q_i having nodes involving unions of fragments, which exist in multiple QPF_i .
 - 2.3 Assign cumulative weights to the nodes depending on their utility to consecutive Q_i (based on frequency and cardinality).
3. Selecting a minimal complete set of AHCPs based on the query weights, subject to storage and maintenance cost.
 - 3.1 For each Q_i , perform top-down evaluation of nodes in its query plan.
 - 3.2 Select lower nodes (backup) if the retrieval cost is lesser.

This part is similar to that of the Algorithm for selecting views to be materialized given an MVPP [YKL97].

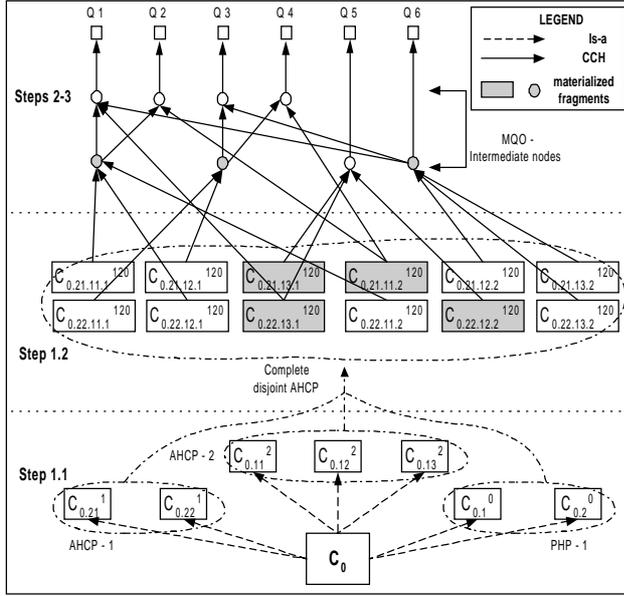


Fig.3. Multiple Partition Processing Plan (MP3)

Figure 3 shows examples of AHCPs (AHCP-1, AHCP-2) and PHPs (PHP-1) on the Fact Class (C_0). These fragments can then be merged by intersecting them to generate a complete disjoint set of Partitions. It must be noted that this is obtained from the query characteristics, and the Partitions are very exhaustive. Due to this reason, it may not be feasible to materialize the fragments all, and hence the MP^3 is used to determine which should be materialized and which should be kept virtual [VLK98]. The cost model is based on the MVPP [YKL97] and incorporates SC, MC and RC. As shown in the figure, the shaded classes are materialized.

4 AHCP evaluation

In this section, we analyze the fragment retrieval cost for processing queries in OQG using AHCP. A comparison of the results with that of plain query processing approach using pointer chasing is then conducted.

4.1 Fragment retrieval cost

In order to evaluate the AHCP methodology, we use the sample ORDW schema and queries as detailed in section 2. Here we note that there are eight queries in the OQG, and we assume them all to be of equal importance.

Running our example through the algorithm given in section 3.3 :

Step 1.1 : For each query Q_i in the OQG, generate logical associated fragments $\{C_{0,j}^p\}$ from $\{C_j^p\}$ that satisfies sub-expressions of Q_i completely.

We see that there are 4 main predicates by which the Dimensions are partitioned, viz. $p1$: "Country = US", $p2a$: "Customer = Teen " , $p2b$: "Customer = Adult", $p3a$: "Product = P1 " , $p3b$: "Product = P2", and $p4$: "Categ = Elec". Performing the AHCP function with respect to these PHPs as shown in section 3.1, we arrive at an exhaustive set of AHCPs of the Sales Class (FC).

Step 1.2 : Perform an intersection of the $C_{0,j}^p$ fragments for all Q_i .

Consequently, by intersection as shown in table 2, we see that a complete set of 16 different AHCPs of the FC (Sales) can be created based on these 4 predicates, encompassing all possible and non-empty fragments :

Table 2. Fragments obtained after intersection

F1	$p1 \wedge p2a \wedge p3a \wedge p4$	F9	$!p1 \wedge p2a \wedge p3a \wedge p4$
F2	$p1 \wedge p2a \wedge p3b \wedge p4$	F10	$!p1 \wedge p2a \wedge p3b \wedge p4$
F3	$p1 \wedge p2a \wedge !p3a \wedge !p3b \wedge p4$	F11	$!p1 \wedge p2a \wedge !p3a \wedge !p3b \wedge p4$
F4	$p1 \wedge p2a \wedge !p4$	F12	$!p1 \wedge p2a \wedge !p4$
F5	$p1 \wedge p2b \wedge p3a \wedge p4$	F13	$!p1 \wedge p2b \wedge p3a \wedge p4$
F6	$p1 \wedge p2b \wedge p3b \wedge p4$	F14	$!p1 \wedge p2b \wedge p3b \wedge p4$
F7	$p1 \wedge p2b \wedge !p3a \wedge !p3b \wedge p4$	F15	$!p1 \wedge p2b \wedge !p3a \wedge !p3b \wedge p4$
F8	$p1 \wedge p2b \wedge !p4$	F16	$!p1 \wedge p2b \wedge !p4$

Step 2.1: For each query Q_i of the OQG, evaluate the minimal set of query processing fragments.

The query processing fragments (QPF) are shown in table 3:

Table 3. Query Processing Fragments (QPF)

QPF_1, QPF_2	F1, F2, F3, F4, F5, F6, F7, F8
QPF_3, QPF_4, QPF_5	F1, F2, F3, F4, F5, F6, F7, F9, F10, F11, F13, F14, F15
QPF_6	F1, F2, F3
QPF_7	F1, F2, F9, F10
QPF_8	F1, F2, F5, F6

Step 2.2 : Create query plans for each Q_i having nodes involving unions of fragments which exist in multiple QPF_i .

The intermediate nodes are created by a combination of fragments noting their affinity in the QPF s. For the sake of completeness, we also create unaccessed nodes as shown in table 4, for example, N12 (F12 U F16), though these fragments are not accessed by

any query in the *OQG*.

Table 4. Intermediate Nodes

Node	Definition	Node	Definition
N1	F1 U F2	N7	N2 U N6 U N9
N2	N1 U F3	N8	F9 U F10
N3	F5 U F6	N9	N1 U N8
N4	N2 U N3	N10	F11 U F13 U F14 U F15
N5	N3 U F7	N11	N5 U N8 U N10
N6	N5 U F4 U F8	N12	F12 U F16

Step 2.3 : Assign cumulative weights to the nodes depending on their utility to consecutive Q_i (based on frequency and cardinality).

For each of the queries Q_i , we know the optimal query processing plan op_i , which is an ordered list of nodes and fragments. We also know the frequency (f_{qi}) of each query, and the selectivity (sel_{pj}) of the clause that its (sub-query) is based on. Depending on those parameters, we give weights to the nodes in the op_i of each query.

For example, processing for Q_1 , we have :

$\langle op_1 \rangle = \langle N7, N6, N2, N9, N3, N5, N8, N1, F1, F2, F4, F5, F6, F7, F8, F9, F10 \rangle$

\therefore the weights for all these nodes (and fragments) is $f_1 * sel_1$.

Processing for Q_6 , we have :

$\langle op_6 \rangle = \langle N2, N1, F1, F2, F3 \rangle$

\therefore the weights for all these nodes (and fragments) is $f_6 * sel_6$.

.. and so on.

For simplification, we consider equal frequencies and 100% selectivity in the fragments, hence at the end of this step, we have weights as shown in table 5:

Table 5. Weights for the Fragments and Nodes

Frag	Weight	Frag	Weight	Node	Weight
F1	4	F11	1	N5	2
F2	4	F12	0	N6	1
F3	2	F13	1	N7	1
F4	1	F14	1	N8	3
F5	3	F15	1	N9	2
F6	3	F16	0	N10	1
F7	1	N1	4	N11	1
F8	1	N2	2	N12	0
F9	3	N3	3		
F10	3	N4	1		

Step 3.1 : For each Q_i , perform top-down evaluation of nodes in its query plan.

As the $\langle op_i \rangle$ are ordered (tree structured), for each Q_i , we can traverse the list in a top-down manner.

Initially all top -level nodes can be considered marked for materialization.

Step 3.2 : Select lower nodes (breakup) if the retrieval cost is lesser.

This is a recursive step, in which the node is unmarked (for materialization) if any node under it has a weight higher than itself. In that case, the lower nodes are considered marked for materialization, and the process is repeated with them.

For example, processing for Q_8 , we mark N4 as it's the first node :

but the weights are : N4 : 1, N2 : 2, N3 : 3.

hence N4 is discarded for N2 and N3.

Now N2 : 2, N1 : 4, F3 : 2.

So N2 is discarded for N1 and F3.

.. and so on.

Repeating this process for all the queries, the following nodes are materialized :

F3, F4, F7, F8, N1, N3, N8, N10.

This is our optimal minimal AHCP set.

4.2 Comparing HCF retrieval cost with pointer traversal cost

In this section, we evaluate our AHCP scheme for its performance gain over the un-partitioned case during query retrieval. As noted in the previous section, we have derived an optimal complete minimal AHCP set of the Sales FC.

The DCs and associated *joins* are in memory and evaluating a query branch dealing with them would involve CPU cost. This is ignored here, as the disk I/O cost is the major component of response time in most query retrieval costs.

The following study shows disk i/o cost ratios for varying relative frequencies of queries in the *OQG*.

$$\text{cost ratio (CR)} = \frac{\text{cost of disk i/o for unpartitioned case}}{\text{cost of disk i/o after AHCP}}$$

The query frequencies are varied from 10% to 90%. As these are relative frequencies, it must be noted that the frequencies of the other queries in *OQG* are modified equally in each case. The parameters for the study are stated in the Appendix.

As can be seen from table 6, there's always a minimum gain obtained when the ORDW is partitioned; the range of the gain varies from 1% to 50% in this case study. Note that these results appear to exhibit a linear relation between the selectivity of the query and the cost gain

obtained from the AHCP operation. However this should be interpreted only as the best-case scenario, because in real-world cases some level of data replication is expected which can cause redundant data access. This may lead to higher cost for the partitioned case than what this example indicates, although the difference will not be too significant.

Table 6. Cost Ratio observations

relative frequency	10%	30%	50%	70%	90%
Q1	0.05	0.15	0.25	0.35	0.45
Q2	0.05	0.15	0.25	0.35	0.45
Q3	0.03	0.09	0.15	0.21	0.27
Q4	0.03	0.09	0.15	0.21	0.27
Q5	0.03	0.09	0.15	0.21	0.27
Q6	0.02	0.06	0.1	0.14	0.18
Q7	0.01	0.03	0.05	0.07	0.09
Q8	0.01	0.03	0.05	0.07	0.09

5 Conclusions and future work

In this paper, we have presented a methodology towards efficient query processing in an object-relational data warehousing (ORDW) environment, through devising and incorporating Associated Horizontal Class Partitioning (AHCP) techniques over the ORDW schema. Our methodology starts with a given set of data warehouse queries, comes up a near-optimal AHCP scheme for the queries, and selects AHCP fragments as *materialized views* to facilitate efficient evaluation of these queries. Through an initial analytical study, we are already able to demonstrate the gains of our approach vis-a-vis the unpartitioned approach in terms of disk I/O in the ORDW environment.

Note that the work we have described in this paper (hence the result obtained) should be only regarded as an intermediate stage towards efficient ORDW query processing; further advanced techniques and mechanisms should and can be naturally added. In particular, an adaptive and extensible indexing framework is currently being developed, so as to better accommodate the requirements of *dynamic data warehousing* [Dayal99] which demands to incorporate more semantics into the data warehouse schemata. As shown in [VLK00], a *query-driven* indexing mechanism built on the SJIH (structural join index hierarchy) [FKL98] seems to be very effective, and is supplementary to the work described in this paper. Moreover, the creation and maintenance algorithms of materialized views and OLAP cubes benefit from the reduced search space obtained due to the AHCP scheme. Since OLAP queries involve multiple paths (multiple selections and group bys), the Forward and Reverse Joins are considerably reduced by employing them on a subset of the AHCP fragments

instead of the entire Fact table. We are currently in the process of combining these complementary approaches into a single framework. We are building an experimental ORDW prototype system that will be validated by empirical studies based on TPC-H benchmark queries.

Acknowledgement - This work has been supported by City University of Hong Kong under grant# 7100078.

References

- [BKS98] L. Bellatreche, K. Karlapalem and A. Simonet, "Algorithms and Support for Horizontal Class Partitioning in Object-Oriented Databases", *Distributed and Parallel Databases*, Kluwer Academic Publishers, accepted in 1998 (to appear).
- [CD97] Surajit Chaudhuri and Umeshwar Dayal, "An Overview of Data Warehousing and OLAP Technology", *ACM SIGMOD Record*, 26(1), March 1997, pp. 65-74.
- [CCS93] E.F. Codd, S.B. Codd, and C.T. Salley, "Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate", Tech. Report, 1993.
- [Dayal99] Umeshwar Dayal, "Dynamic Data Warehousing", *Proc. First International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, Florence, Italy, 1999.
- [Fun98] Chi-wai Fung, "Vertical Class Partitioning and Complex Object Retrieval in Object Oriented Databases", Ph.D. Thesis, Department of Computer Science, HKUST, Dec 1998.
- [FKL98] Chi-wai Fung, Kamalakar Karlapalem, Qing Li, "Structural Join Index Hierarchy: A Mechanism for Efficient Complex Object Retrieval", *Proc. FODO Conference 1998*, pp. 127-136.
- [KL00] K. Karlapalem and Q. Li, "A Framework for Class Partitioning in Object-Oriented Databases", *Distributed and Parallel Databases*, 8(3):317-350, Kluwer Academic Publishers, 2000 (in press).
- [GGT96] G. Gardarin, J.R. Gruser, Z.H. Tang, "Cost-based Selection of Path Expression Processing Algorithms in Object-Oriented Databases", *Proc. VLDB 1996*, pp. 390-401.
- [GHRU97] H. Gupta, V. Harinarayanan, A. Rajaraman, and J.D. Ullman, "Index Selection for OLAP", *Proc. ICDE 1997*, pp. 208-219.
- [GM95] A. Gupta, and I. S. Mumick, "Maintenance of Materialized Views: Problems, Techniques, and Applications", *IEEE Data Eng. Bulletin*, June 1995.

- [MK99] Mukesh Mohania and Y. Kambayashi, "Making Aggregate Views Self-Maintainable", *Data and Knowledge Engineering*, accepted in 1999 (to appear).
- [OQ97] P. O'Neil, D. Quass, "Improved query performance with variant indexes", *Proc. ACM SIGMOD '97*, pp. 38-49.
- [Rou97] Nick Roussopoulos, "Materialized Views and Data Warehouses", *Proc. KRDB 1997*, pp. 12.1-12.6.
- [Sar97] Sunita Sarawagi, "Indexing OLAP Data", *IEEE Data Engineering Bulletin*, 1997, 20:36-43.
- [VLK98] Vivekanand Gopalkrishnan, Qing Li, Kamalakar Karlapalem, "Issues of Object-Relational View Design in Data Warehousing Environment", *Proc. IEEE SMC Conference 1998*, pp. 2732-2737.
- [VLK99] Vivekanand Gopalkrishnan, Qing Li, Kamalakar Karlapalem, "Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies", *Proc. First Int'l Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, LNCS 1676, pp. 11-22, Florence, Italy, 1999.
- [VLK00] Vivekanand Gopalkrishnan, Qing Li, Kamalakar Karlapalem, "Efficient Query Processing with Structural Join Indexing in an Object Relational Data Warehousing Environment", *Proc. 11th Information Resources Management Association Int'l Conference (IRMA'00)*, Anchorage, Alaska, May 21-24, 2000 (to appear).
- [YKL97] Jian Yang, Kamalakar Karlapalem, Qing Li, "Algorithms for Materialized View Design in Data Warehousing Environment", *Proc. VLDB 1997*, pp. 136-145.

Appendix

Table A. Query Parameters

f_o = fan-out
R - reference (reverse links)
 $\|C_i\|$ - cardinality

Reference (i→j)	f_o	R	$\ C_i\ $	$\ C_j\ $
Sales→Product	1	100	50M	.5M
Sales→Customer	1	50	50M	1M
Sales→Teenager	1	250	50M	.2M
Sales→Date	1	500	50M	36.5K
Prod→Category	1	10	.5M	1K
Product→Retailer	50	100	.5M	50K
Category→Type	100	5	1000	10
Retailer→City	1	4	50K	12.5K
Customer→City	1	80	1M	12.5K
Year→Mon	12	1	10	120
Mon→Date	30	1	120	3.6K
Year→Date	365	1	10	3.6K
Country→State	25	1	10	250
State→City	5	1	250	1.2K
Country→City	125	1	10	1.2K

Table B. Selectivity (%) :

Country = 'US'	50
Category = 'Elec'	30
Product = 'P1'	5
Product = 'P2'	5
Customer = 'Teen'	20