Achieving Scalability in OLAP Materialized View Selection

Thomas P. Nadeau
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan
nadeau@engin.umich.edu

Toby J. Teorey
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan
teorey@eecs.umich.edu

ABSTRACT

The goal of on-line analytical processing (OLAP) is to quickly answer queries from large amounts of data residing in a data warehouse. Materialized view selection is an optimization problem encountered in OLAP systems. Published work on the problem of materialized view selection presents solutions scalable in the number of possible views. However, the number of possible views is exponential relative to the number of database dimensions. A truly scalable solution must be polynomial time relative to the number of dimensions. We present such a solution, our Polynomial Greedy Algorithm. Complexity analysis proves scalability, and a performance study verifies the result. Empirical evidence demonstrates benefits close to existing algorithms. We conclude the Polynomial Greedy Algorithm functions effectively where existing algorithms fail dramatically.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design – *data models, schema and subschema.*

General Terms

Algorithms, Performance, Design, Theory.

Keywords

OLAP, materialized views, view selection, data warehouse, OLAP performance.

1. INTRODUCTION

Accumulation of data in industry and organizations has led to large archives of data in recent years. Quick access to the information in these archives has become critical for decision-making. The need to excel has given rise to new data models and decision support systems. Typically the queries posed involve operations of aggregation such as sum or count. The queries also typically include "group by" expressions. For example, the CEO of a book manufacturing company may want to examine trends in profitability of different types of books over time. The answer could be found by doing a sum of the cost and sell values of jobs, grouped by bind style and quarter. Data warehouses have been engineered to answer queries of aggregation with "group by" expressions efficiently. The goal of on-line analytical processing (OLAP) is to quickly answer queries from large amounts of data residing in a data warehouse.

1.1 OLAP: The beast of burden

Data warehouses are commonly organized with one large central fact table, and many smaller dimension tables. This configuration is termed a star schema. Figure 1 illustrates with an example. The fact table is composed of two types of attributes: dimension attributes and measures. The dimension attributes in Figure 1 are *CustID*, *DateID* and *BindID*. The dimension attributes have foreign-key/primary-key relationships with the dimension tables. Together, the dimension attributes compose the primary key of the fact table. The dimension attributes are typically used in query "group by" expressions, or to join the fact table with the dimension tables. Notice that a dimension can also have a hierarchy. For example, the Figure 1 schema allows time to be grouped by *DateID*, *Month*, *Quarter* or *Year*. The fact table also contains measure attributes, the values to be aggregated. The measure attributes in Figure 1 are *Cost* and *Sell*.

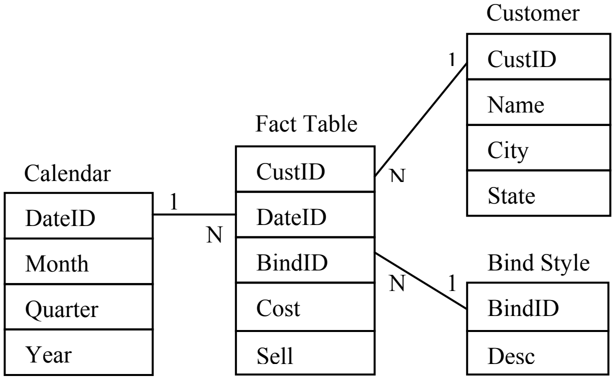

Figure 1. Example star schema for a data warehouse

A fact table in a data warehouse may contain many millions of rows, and processing a single query can require significant resources. To improve the quickness of response to queries, pre-aggregation is a useful OLAP strategy. Pre-aggregation requires the result to be saved to disk as materialized views, also known as automatic summary tables (ASTs). The number of possible views is exponential in the number of dimensions in the database. Faced with combinatorial explosion, limited disk space and limited update resources, OLAP must select a strategic set of beneficial views to materialize in order to achieve quick response to queries.

DOLAP '02, November 8, 2002, McLean, Virginia, USA.
Copyright 2002 ACM 1-58113-590-4/02/0011…$5.00.

1.2 The map to quick response

The goal of OLAP is to give quick response to queries posed against a large repository of data. Figure 2 maps our approach to the general problem of OLAP optimization. We break the larger problem of OLAP optimization into four sub-problems: *View Size Estimation*, materialized *View Selection*, materialized *View Maintenance*, and *Query Optimization* with materialized views. We explain this diagram, setting the context of our work in view selection.

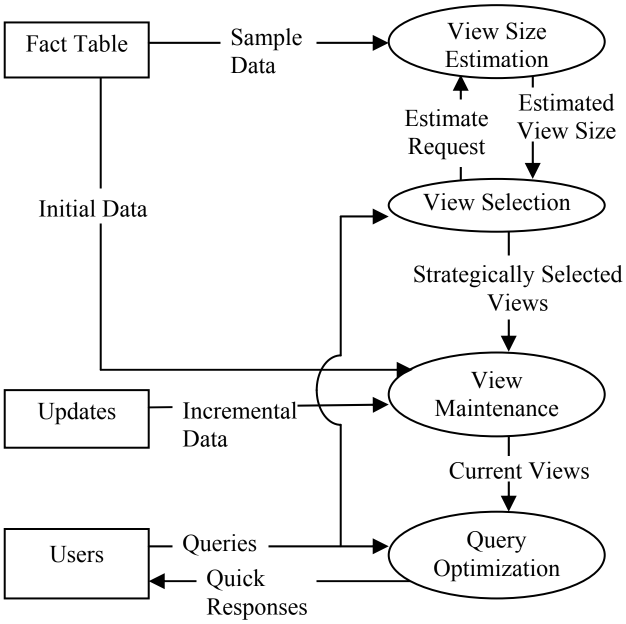

Figure 2. Our plan for OLAP optimization

Our plan for OLAP optimization feeds *Sample Data* from the *Fact Table* into *View Size Estimation*. *View Selection* makes an *Estimate Request* for the view size of each new view it encounters while exploring promising regions of the search space. *View Size Estimation* queries the *Sample Data*, examines and models the distribution [12]. The distribution observed in the sample is used to estimate the expected number of rows in the view for the full dataset. The *Estimated View Size* is passed to *View Selection*, which uses the estimates to evaluate the relative benefits of materializing the various views under consideration. *View Selection* picks *Strategically Selected Views* for materialization with the goal of minimizing total query costs. *View Maintenance* builds the original views from the *Initial Data* from the *Fact Table*, and maintains the views as *Incremental Data* arrives from *Updates*, offering *Current Views* for use by *Query Optimization*. *Query Optimization* must consider which of the *Current Views* can be utilized to most efficiently answer *Queries* from *Users*, giving *Quick Responses* to the *Users*. *Queries* feed back into *View Selection*, allowing the system to be dynamic, adapting and improving over time, matching the query workload.

1.3 Organizing the expedition

The focus of this paper is the view selection algorithm. We describe a new view selection algorithm scalable relative to the number of dimensions present in the data warehouse. This dimensional scalability is a quantum leap over existing view selection algorithms that search the hypercube lattice structure first described by Harinarayan et al. [6]. We demonstrate that our Polynomial Greedy Algorithm (PGA) selects sets of views with close to the same benefits as the greedy algorithm [6] (we refer to their algorithm henceforth as HRU). Furthermore, we demonstrate PGA functions beyond where HRU is overwhelmed by the exponential explosion of the possible views to analyze.

We focus on comparing PGA with HRU since most research in materialized view selection builds upon Harinarayan et al. [6]. The algorithms presented in the related research [2, 4, 5, 14, 16] also suffer from the problem of exponential explosion with scaled up dimensionality, since they all potentially process every node in the hypercube lattice structure. A survey of related works is included with the extended version [13]. A comparison with the straight forward HRU approach demonstrates our purpose. Extensions accounting for query frequencies, index structure and update costs are described in future work.

The problem of exponential explosion that exists in previous work is discussed in Section 2. A PGA example is given in Section 3. Section 4 presents our solution to overcoming the exponential explosion. Section 5 verifies the usefulness of our algorithm with empirical test results. Conclusions are expressed in Section 6. Section 7 outlines future work.

2. EXPONENTIAL EXPLOSION

Most of the previous work in view selection [2, 3, 4, 5, 14, 16] is based on the hypercube lattice structure [6]. We briefly review the concepts associated with the hypercube lattice structure, and HRU. Then we point out the scalability issue that has not been previously addressed.

Figure 3 illustrates the hypercube lattice structure with an example [6]. Each node of the lattice structure represents a possible view. Each node is labeled with the set of dimensions in the "group by" list for that view. The numbers associated with the nodes represent the number of rows in the view. These numbers are normally derived from a view size estimation algorithm. However, the numbers in Figure 3 follow the example as given by Harinarayan et al. [6]. The root node in the lattice structure represents the fact table. A given view can be calculated from any materialized ancestor view. The initial state for HRU has only the fact table materialized. HRU calculates the benefits of each possible view during each iteration, and selects the most beneficial view for materialization. Processing continues until a predetermined number of materialized views is been reached.

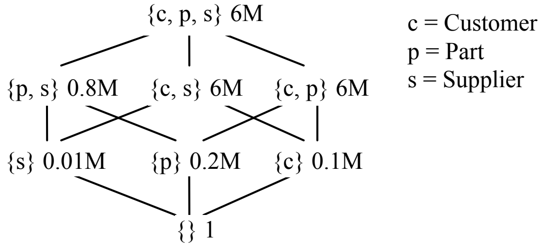

Figure 3. Example hypercube lattice structure [6]

Table 1 shows the calculations for the first two iterations of HRU. Materializing {p, s} saves 6M – 0.8M = 5.2M rows for each of 4 views ({p, s} and its three descendants: {p}, {s} and {}). The view {c, s} yields no benefit if materialized, since any query that can be answered by reading 6M rows from {c, s} can also be answered by reading 6M rows from the fact table {c, p, s}. The view {p, s} is selected for materialization in the first iteration. {c} is selected in the second iteration.

Table 1. Two iterations of HRU, based on figure 3

	Iteration 1 Benefit	**Iteration 2 Benefit**
{p, s}	**5.2M x 4 = 20.8M**	
{c, s}	0 x 4 = 0	0 x 2 = 0
{c, p}	0 x 4 = 0	0 x 2 = 0
{s}	5.99M x 2 = 11.98M	0.79M x 2 = 1.58M
{p}	5.8M x 2 = 11.6M	0.6M x 2 = 1.2M
{c}	5.9M x 2 = 11.8M	**5.9M x 2 = 11.8M**
{}	6M - 1	0.8M - 1

This simple example is sufficient to illustrate a crucial point. The algorithm evaluates every unselected node during each iteration, and each evaluation considers the effect on every descendant. The algorithm consumes $O(kn^2)$ time where k = |views to select| and n = |nodes|. This order of complexity looks very good, it is polynomial time. However, the result is misleading. The number of nodes in the lattice structure is exponential relative to the number of dimensions. Consider a database with no hierarchies. When we pose a query to OLAP, for each dimension we can choose whether or not to group by that dimension. The number of possible views is therefore 2^d where d = |dimensions|. Thus $n = 2^d$, and the time complexity of HRU is $O(k2^{2d})$. HRU runs in time exponential relative to the number of dimensions in the database.

Any algorithm that considers every node in the hypercube lattice structure is exponential relative to the number of dimensions. All the published static view selection algorithms we are aware of potentially touch every node in the lattice. The exponential time complexity swamps any processor as the number of dimensions is scaled up. Let us drive this home with an example. We implemented HRU and tested the algorithm at ten dimensions, with no hierarchies. The associated lattice has $2^{10} = 1024$ nodes. The algorithm selected 120 views in 1 hour. Now let's say the CEO wants the option to choose from 20 job attributes instead of 10. The number of possible views just jumped 1024 fold. The view selection algorithm now takes over one million hours to select 120 views. Clearly the CEO would find this unacceptable! We must find a better solution.

3. PGA EXAMPLE

We walk through the example in Figure 3 using PGA, and discuss the theory behind our design in Section 4. PGA, like HRU, also selects one view for materialization with each iteration. However, PGA divides each iteration into a nomination phase and a selection phase. The first phase nominates promising views into a candidate set. The second phase estimates the benefits of materializing each candidate, and selects the view with the highest evaluation for materialization. Figure 4 outlines PGA activities. A more detailed activity diagram is available in the extended version [13].

The nomination phase begins at the top of the lattice; in Figure 3, this is the node {c, p, s}. We nominate the smallest node from amongst the children. The candidate set is now {{p, s}}. We then examine the children of {p, s} and nominate the smallest child, {s}. The process repeats until the bottom of the lattice is reached. The candidate set is then {{p, s}, {s}, {}}. Once a path of candidate views has been nominated, the algorithm enters the selection phase.

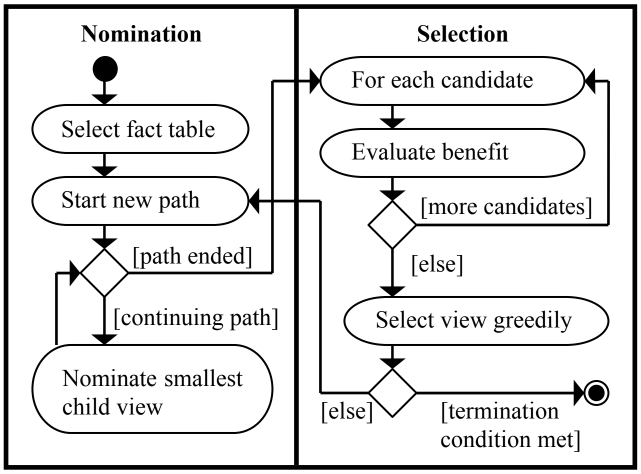

Figure 4. Activity diagram of PGA

The selection phase evaluates each view in the candidate set, and selects the view that appears to yield the most benefit if materialized. The evaluation for each candidate is done by taking the difference with the smallest ancestor selected for materialization, and multiplying the savings by the estimated number of nodes affected. The number of nodes affected is estimated in two steps. The number of descendants is calculated, including the candidate itself. Then we look for materialized views smaller than the candidate. If any such view is found, we account for the affect of the view with the largest number of descendants in common with the candidate. The overlapping descendants are subtracted from the count of views affected by materializing the candidate, since the benefits are undercut by the other view. The example in Figure 3 is processed as shown in Table 2 and Table 3.

Table 2. First iteration of PGA, based on figure 3

Candidates	**Iteration 1 Benefit**
{p, s}	**5.2M x 4 = 20.8M**
{s}	5.99M x 2 = 11.98M
{}	6M - 1

Table 3. Second iteration of PGA, based on figure 3

Candidates	**Iteration 2 Benefit**
{c, s}	0 x 2 = 0
{s}	0.79M x 2 = 1.58M
{c}	**5.9M x 2 = 11.8M**
{}	6M - 1

The view {p, s} is selected for materialization during the first iteration, and is removed from the candidates. Then the second iteration begins with another nomination phase. The algorithm examines the children of {c, p, s} and nominates the smallest

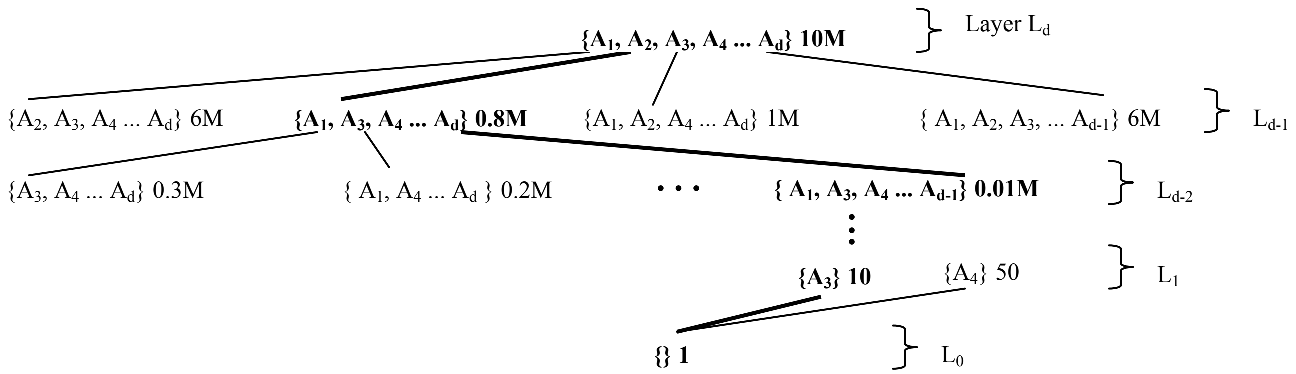

Figure 5. Example path of candidate views

child that has not already been nominated. There is a tie in this instance between {c, s} and {c, p}. Ties are broken arbitrarily; we nominate the view {c, s}. Then the children of {c, s} are examined and the smallest child that has not been previously nominated, {c}, is nominated. Then the candidates are evaluated as shown in Table 3. The view {c} is selected for materialization.

Compare Tables 2 and 3 with Table 1. Notice our algorithm does fewer calculations than HRU, and yet reaches the same decisions in this example as HRU. Our algorithm is polynomial time in the number of dimensions, whereas HRU is exponential in the number of dimensions. We discuss the advantages of a polynomial time algorithm in Section 4. We demonstrate the advantages and quality empirically in Section 5.

4. TAMING DIMENSIONALITY

There are two sources of exponential time complexity in the algorithms of Harinarayan et al. [6], and the enhanced algorithms [2, 4, 5, 14, 16]. First, each iteration evaluates every node (view) in the hypercube lattice structure. The second source of exponential time complexity is that each node evaluation considers the effect of materialization on every descendant. We present our approach to overcoming each of these complexity barriers in turn.

4.1 Nominating views in polynomial time

The key to overcoming the barrier of evaluating an exponential number of nodes is to consider only promising portions of the lattice. Figure 5 illustrates with an example how we identify promising portions of the lattice in polynomial time. Candidate views are nominated by following the steepest path through the lattice. The nodes in bold face are the candidate views. Only a small portion of the lattice is shown. There is no need to generate the entire lattice. We represent each view with a fixed amount of metadata. This greedy approach for selecting a path is polynomial in both time and space relative to d, the number of dimensions.

When evaluating views on the same level of the lattice, smaller views are typically better to materialize than larger views, since smaller views tend to yield greater benefits at smaller costs. This is illustrated by the example in Section 2. Our approach is to find paths of nodes through the lattice that appear promising for high benefits.

Our approach first nominates promising views as candidates for materialization. These candidate views are later evaluated for view selection. The nomination process begins with the root as the parent node, which is the fact table. We determine the identities of the children for that node. The view size is estimated for each child. The smallest child view is nominated as a candidate for view materialization. The nomination process then moves down one level. The nominated child is considered as a parent, and the smallest child of that node is nominated as a candidate view. The process of nominating candidate views continues until the bottom of the lattice is reached. This nominating process constitutes a greedy search following the steepest path through the lattice. The result is a path of candidate views that promise high benefits.

There are a few properties of the hypercube lattice that are noteworthy for analyzing the complexity of the nomination process. Let the fact table have d dimensions. Let the bottom layer of the lattice be labeled L_0, the next layer up be labeled L_1 and so on through L_d. The root node of the lattice has d dimensions. Each node of layer L_{d-1} has d – 1 dimensions, since each of these views represents aggregation along one dimension of the parent view. There are d nodes in L_{d-1}, since there are d possible dimensions available for aggregation in the parent node. If we now consider a node from L_{d-1} as a parent, this node is the root of a hypercube with d – 1 dimensions, formed by its descendants. The pattern repeats with one less dimension. In general, a node in level L_i has i dimensions, and i children. There are d levels in the larger structure, not counting the fact table. The swatch of nodes considered when nominating a path of candidate views is widest at L_{d-1} with d nodes. The swatch cuts through d levels. Thus the time complexity for nominating a path of candidate views is $O(d^2)$.

Our nomination process permits the implementation of a polynomial time algorithm relative to the number of dimensions in the database. Our process does not explore the search space as broadly as HRU, but never-the-less in practice finds a set of views to materialize that is nearly as beneficial as the set selected by HRU. There are several reasons for the success with relatively little processing. The candidate set is composed of the smallest views among siblings. There is a very strong correlation between the size of the view and the benefit of materializing the view. The smaller views tend to yield a greater difference in size compared to its smallest materialized ancestor, which leads to greater I/O savings whenever the view is utilized at query time.

Our nomination strategy begins at the top of the lattice and moves downward. There is a tendency for views high up the lattice structure to be selected for materialization, because the savings generated by the difference with the smallest materialized ancestor is multiplied by a larger set of descendant views. Beginning the nomination process at the top helps assure the best views high up the lattice are found and considered for materialization. The nomination process cuts vertical paths through the lattice structure. Although there is a tendency for selected views to reside high in the lattice, it is possible for unusually small views lower down in the lattice to yield great benefit. There is a tendency for small parent views to have small children views. The strategy of following generations of small views through the lattice structure helps assure that unusually small views low in the lattice are found and considered for materialization.

PGA alternates between nomination and selection phases. Promising views in a path are nominated as candidates. Then a view is selected from the candidates for materialization. The process iterates between nomination and selection phases until a predetermined termination condition is met (e.g. a fixed amount of disk space).

4.2 Selecting views in polynomial time

The second barrier of exponential processing can be overcome by approximating the benefits derived by materialization. PGA estimates benefits by taking the difference between the size of view V, and its smallest selected ancestor. PGA also takes into account the selected view that most prominently affects the number of nodes benefiting from the materialization of V. If a selected view is smaller or equal in size to V, then any common descendants do not benefit from materializing V. These calculations only take into account very prominent interactions with selected views, yet the result is surprisingly useful. This strategy constitutes the materialization benefit estimator we implemented in PGA.

We track metadata on the candidate views to reduce the time complexity of the algorithm. For each candidate, we store the smallest selected ancestor rows, and the maximum number of descendants in common with any selected view. Tracking this metadata allows the *Evaluate benefit* activity in Figure 4 to execute once in O(1) time. The algorithm is dominated by the process of setting the metadata. The metadata is updated when a view is nominated or selected. Given two views, the number of common descendants can be determined in O(d) time. When a view is nominated, the number of common descendants are computed for each of O(k) selected views. The *Nominate smallest child view* activity of Figure 4 executes O(d) times during each *Nomination* phase. Thus one *Nomination* phase executes in $O(d^2k)$ time. The *Select view greedily* activity computes the common descendants for each of O(dk) candidate views. Thus the *Selection* phase also executes in $O(d^2k)$ time. Since k views are selected, the overall time complexity of PGA is $O(d^2k^2)$.

A fixed amount of metadata is stored for each of $O(d^2)$ views for each of O(k) paths through the lattice. Thus the space complexity of PGA is $O(d^2k)$.

When hierarchies are present, the lattice has more layers. Let h_i be the number of hierarchical levels in dimension i.

$$\text{Lattice layers } \ell = 1 \text{ fact table} + \sum_{i=1}^{d} (h_i - 1) \text{ aggregation layers}$$

PGA complexity is $O(dk^2\ell)$ time and $O(dk\ell)$ space.

A more accurate benefit estimate could be obtained by accounting for the effects of common descendants between a candidate and each materialized view. Considering all pair-wise interactions with materialized views would increase accuracy and processing time. This approach would still not consider interactions of three or more materialized views.

There is a progression of possible polynomial time node evaluation algorithms. We could account for three-node interactions. Then we could account for four-node interactions and so on. Each level of detail adds to the complexity of evaluation, with potential benefits in the resulting query response times. We find PGA yields effective results.

5. EMPIRICAL EVIDENCE

Karloff and Mihail [9] point out that no performance guarantee relative to the optimal solution has been proven for any existing algorithm that is polynomial time relative to the number of views. Following the advice of Karloff and Mihail [9], we demonstrate the worthiness of PGA with empirical evidence.

The test environment is a Pentium IV 1.9 GHz processor, with 1GB RAM, running Windows 2000. The programs are written in Microsoft Visual Basic. Data is stored using Microsoft SQL Server 7.0. The databases contain real data obtained from a local book manufacturing company, McNaughton & Gunn, Inc.

Figures 6 through 12 are the results of tests carried out over hierarchical databases. The query costs shown are the number of rows retrieved from disk if every possible view is queried exactly once, as done by Harinarayan et al. [6]. The exact number of rows for each view is pre-calculated for these experiments. Normally these numbers would derive from a view size estimator. Using the exact number of rows isolates deviations caused by view selection from deviations caused by view size estimation. The differences in query costs are strictly the result of the differences in the view selection algorithms.

The dimensions in the datasets all have four levels in their hierarchies. More detailed descriptions of the database schemas are available in the extended version [13]. Figure 6 illustrates the relative accuracy of PGA compared with HRU and the optimal solution at small dimensionality. The scale-up into higher dimensionality is demonstrated in Figures 7 through 12.

Figures 6 through 9 plot the query costs achieved at various levels of view materialization. The trend is similar to that of a cache; the more cache, the quicker the general response, with diminishing returns. Likewise, the more materialized views, the quicker the response, with diminishing returns. Figure 6 compares query costs over a two dimensional hierarchical data set, four levels per dimension. There are 4^2 or 16 possible views. Three nearly identical curves are shown. The key point is that PGA achieves very close to the same query response as HRU and the optimal solution.

The optimal solution can only be calculated at very small dimensionalities. Processing the optimal solution quickly becomes untenable at four dimensions. Under these conditions, we are still able to demonstrate our algorithm derives benefits close to HRU as shown in Figures 7 and 8.

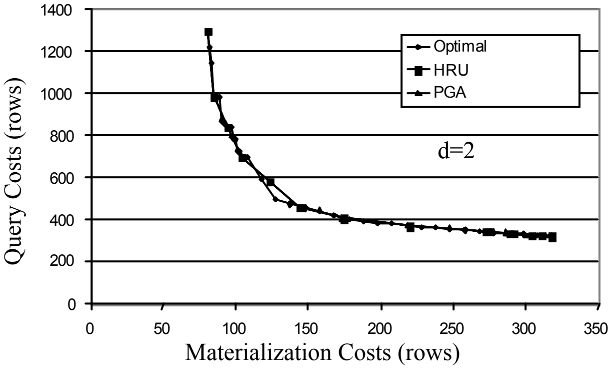

Figure 6. Query costs at two dimensions

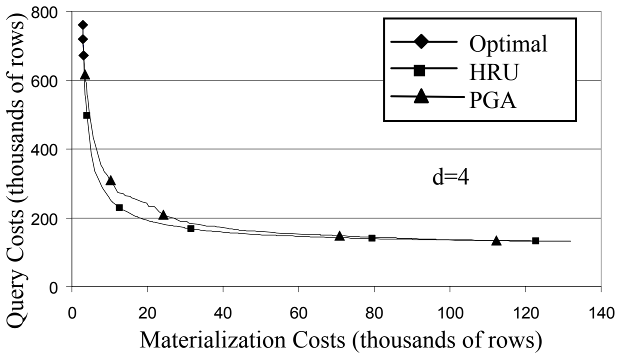

Figure 7. Query costs at four dimensions

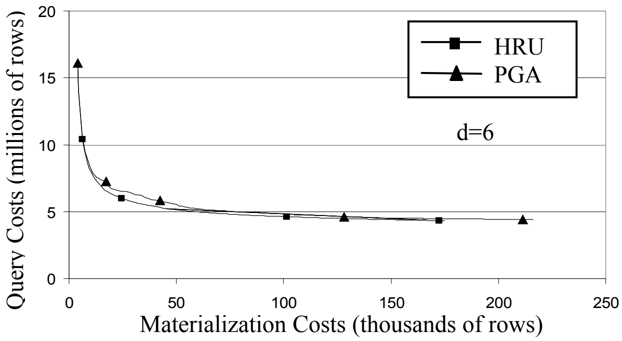

Figure 8. Query costs at six dimensions

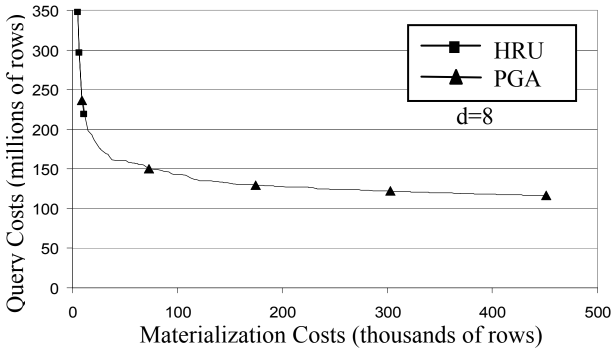

Figure 9. Query costs at eight dimensions

PGA and HRU tend to agree for the most critical early decisions. The first four views selected are identical in the six dimensional experiment. Out of the first 128 views selected, 45% are common between PGA and HRU. The divergence is not surprising, since HRU is also a greedy approach. Different heuristics can result in different decisions. PGA averages only 4.9% higher than HRU in query costs at six dimensions.

The processing times for view selection are shown in Figures 10, 11 and 12. The number of possible views is 4^4, 4^6, 4^8 respectively at four, six and eight dimensions. The exponential complexity of HRU swamps the processor as we scale up. HRU becomes completely untenable at eight dimensions, as shown in Figure 12. PGA continues to find good sets of materialized views when HRU has failed.

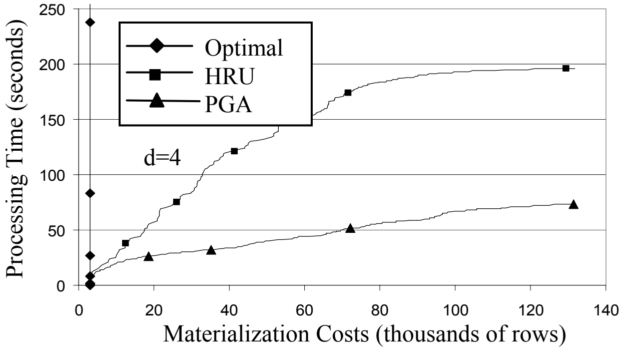

Figure 10. View selection processing at four dimensions

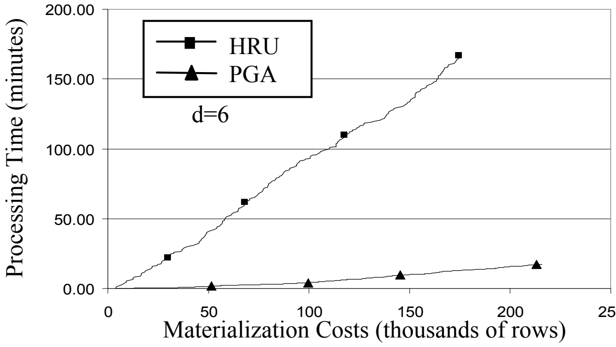

Figure 11. View selection processing at six dimensions

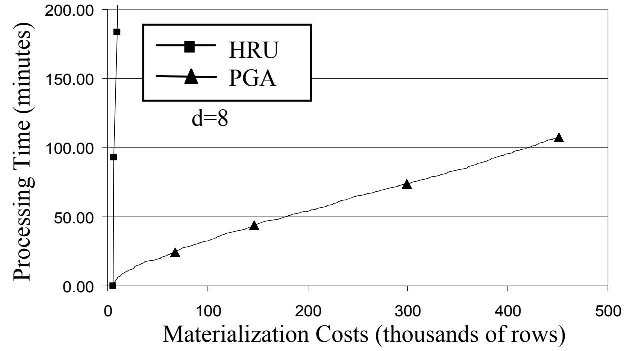

Figure 12. View selection processing at eight dimensions

6. SUMMARY: ENJOYING THE VIEWS

The complexity of PGA is $O(dk^2\ell)$ time and $O(dk\ell)$ space where d is the number of dimensions, k is the number of views selected for materialization, and ℓ is the number of layers in the lattice. This compares with $O(k2^{2d})$ time complexity for HRU. Figures 6 through 12 illustrate the value of having a view selection algorithm that is polynomial time relative to the number of dimensions. The optimal solution can realistically be found only for small dimensionality. Figure 6 shows both HRU and PGA come close to the optimal solution. The scale-up testing begins with Figure 7. With a four dimensional hierarchical database, it is no longer possible to find the optimal solution. Figure 10 shows the processing time required for the optimal solution is nearly a vertical line. The HRU algorithm is more scaleable than finding the optimal solution, but begins to fail for a six dimensional hierarchical database (see Figure 11), and fails miserably with an eight dimensional database (see Figure 12). HRU consumes 91 minutes for each view selection for in this environment, whereas PGA processes 256 view selections in 108 minutes for an average of 1 every 25 seconds. PGA succeeds in finding a good set of materialized views where HRU can no longer function.

There is a general principle that could be implemented in a commercial system. There is a tradeoff between processing view selection, and the resulting query costs. An OLAP system could calculate the processing time required for various approaches, based on the complexity of the view selection algorithms, and use the best feasible approach given the situation. For small dimensional databases, it may actually be possible to determine the optimal materialized view selection. When it is not possible to find the optimal solution, it may be possible to use HRU. When HRU fails, then PGA extends the usefulness of the OLAP system.

7. THE FUTURE

PGA can be easily adapted to account for query frequencies if known in advance. The views that would naturally answer these queries should be added to the candidate set at the start. This permits the possibility of materializing commonly utilized views. View evaluation needs to query metadata to find frequently accessed views that are descendants of the evaluated view, and multiply benefits at the descendant view by the query frequency.

Accounting for update costs would be a matter of modifying the metric associated with the views. Replacing rows with disk accesses as done in Uchiyama et al. [16] would express query and update costs in common units, allowing for minimization of disk I/O. The search mechanism would remain unchanged.

Index structures should also be figured into the selection of materialized views. Implementation is dependent in part on the index mechanisms. We are examining multi-dimensional index structures along the lines of cubetrees [10].

We are exploring alternative data structures for storing and accessing materialized views in OLAP, and the implications in terms of query response and update costs. We hope to contribute to research on fast materialized view updates. Our ultimate goal is to contribute a new integrated OLAP optimization approach.

8. REFERENCES

[1] Agarwal, S., Agrawal, R., Deshpande, P., Gupta, A., Naughton, J. F., Ramakrishnan, R., Sarawagi, S. On the Computation of Multidimensional Aggregates. In Proceedings of VLDB '96 (Mumbai, India, September 1996), Morgan Kaufman, 506-521.

[2] Baralis, E., Paraboschi, S., Teniente, E. Materialized Views Selection in a Multidimensional Database. In Proceedings of VLDB '97 (Athens, Greece, August 1997), Morgan Kaufman, 156-165.

[3] Gupta, H. Selection of Views to Materialize in a Data Warehouse. In Proceedings of ICDT '97 (Delphi, Greece, January 1997), Springer, 98-112.

[4] Gupta, H., Harinarayan, V., Rajaraman, A., Ullman, J. D. Index Selection for OLAP. In Proceedings of ICDE '97 (Manchester, UK, April 1997), IEEE Computer Society Press, 208-219.

[5] Gupta, H., Mumick, I. S. Selection of Views to Materialize Under a Maintenance Cost Constraint. In Proceedings of ICDT '99 (Jerusalem, Israel, January 1999), Springer, 453-470.

[6] Harinarayan, V., Rajaraman, A., Ullman, J. D. Implementing Data Cubes Efficiently. In Proceedings of SIGMOD '96. (Montreal, Canada, June 1996), ACM Press, 205 - 216.

[7] Janakiraman, J., Warack, C., Bhal, G., Teorey, T. J. Progressive Fragment Allocation. In Proceedings of ER '91 (San Mateo CA, USA, 1991), ER Institute, 543-560.

[8] Kimball, R. The Data Warehouse Toolkit. John Wiley & Sons, 1996.

[9] Karloff, H. J., Mihail, M. On the Complexity of the View-Selection Problem. In Proceedings of PODS '99 (Philadelphia PA, USA, June 1999), ACM Press, 167 - 173.

[10] Kotidis, Y., Roussopoulos, N. An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. In Proceedings of SIGMOD '98 (Seattle WA, USA, June 1998), ACM Press, 249-258.

[11] Kotidis, Y., Roussopoulos, N. DynaMat: A Dynamic View Management System for Data Warehouses. In Proceedings of SIGMOD '99 (Philadelphia PA, USA, June 1999), ACM Press, 371-382.

[12] Nadeau, T. P., Teorey, T. J. A Pareto Model for OLAP View Size Estimation. In Proceedings of CASCON '01 (Toronto, Canada, November 2001), IBM Canada, 1 – 13.

[13] Nadeau, T. P., Teorey, T. J. Achieving Scalability in OLAP Materialized View Selection. Technical Report (extended version). http://www.eecs.umich.edu/~teorey/cv.html.

[14] Shukla, A., Deshpande, P., Naughton, J. F. Materialized View Selection for Multidimensional Datasets. In Proceedings of VLDB '98 (New York NY, USA, August 1998), Morgan Kaufman, 488-499.

[15] Thomsen, E. OLAP Solutions. John Wiley & Sons, 1997.

[16] Uchiyama, H, Runapongsa, K., Teorey, T. J. Progressive View Materialization Algorithm. In Proceedings of DOLAP '99 (Kansas City MO, USA, 1999), 36 – 41.