Automated design of multidimensional clustering tables for relational databases

Sam S. Lightstone
IBM Toronto Laboratory
Markham, Ontario, Canada
light@ca.ibm.com

Bishwaranjan Bhattacharjee
IBM T. J. Watson Research Center
Hawthorne, New York, USA
bhatta@us.ibm.com

Abstract

The ability to physically cluster a database table on multiple dimensions is a powerful technique that offers significant performance benefits in many OLAP, warehousing, and decision-support systems. An industrial implementation of this technique for the DB2® Universal Database™ (DB2 UDB) product, called multidimensional clustering (MDC), which co-exists with other classical forms of data storage and indexing methods, was described in VLDB 2003. This paper describes the first published model for automating the selection of clustering keys in single-dimensional and multidimensional relational databases that use a cell/block storage structure for MDC. For any significant dimensionality (3 or more), the possible solution space is combinatorially complex. The automated MDC design model is based on what-if query cost modeling, data sampling, and a search algorithm for evaluating a large constellation of possible combinations. The model is effective at trading the benefits of potential combinations of clustering keys against data sparsity and performance. It also effectively selects the granularity at which dimensions should be used for clustering (such as week of year versus month of year). We show results from experiments indicating that the model provides design recommendations of comparable quality to those made by human experts. The model has been implemented in the IBM® DB2 UDB for Linux®, UNIX® and Windows® Version 8.2 release.

**Proceedings of the 30th VLDB Conference,
Toronto, Canada, 2004**

1. Introduction

Multidimensional clustering (MDC) techniques have been shown to have very significant performance benefits for complex workloads [4][12][14][15][20]. In fact, the literature on MDC has focused on how to better design database storage structures, rather than on how to select the clustering dimensions. However, for any given storage structure used for MDC, there are complex design trade-offs in the selection of the clustering dimensions. In this paper we present a model for doing so in the form of an MDC Advisor that will select MDC keys (i.e., designs) optimized for a specified combination of workload, schema, and data. We also describe its implementation for the MDC physical layout scheme introduced in DB2 UDB Version 8.1 [2] and report the results of experiments that indicate the model provides design recommendations that are in line with the quality of human expert recommendations. The value of exploiting MDC would be superior system performance, reduced time from test to production system, and reduced skill requirements within an enterprise.

MDC is motivated to a large extent by the spectacular growth of relational data, which has spurred the continual research and development of improved techniques for handling large data sets and complex queries. In particular, online analytical processing (OLAP) and decision-support systems (DSS) have become popular for data mining and business analysis [16]. OLAP and DSS systems are characterized by multidimensional analysis of compiled enterprise data, and typically include transactional queries including group-by, aggregation, (multidimensional) range queries, cube, roll-up and drill-down.

The performance of multidimensional queries, (such as GROUP BY and range queries) is often improved through data clustering, which can significantly reduce I/O costs, and modestly reduce CPU costs. Yet the choice of clustering dimensions and the granularity of the clustering are nontrivial choices and can be difficult to design even for experienced database designers and industry experts.

In recent years, there have been several research and industrial initiatives focused on physical database design. In particular, a number of projects have focused on design automation for indexes, materialized views, and table partitioning [3][5][6][7][8][13][17][18][19].

The recent flurry of papers on index and materialized view selection, and the development of industrial applications in self-managing, or autonomic, systems by leading RDBMS vendors such as Microsoft, IBM and Oracle, all attest to the growing corporate recognition of this important area of investigation.

The rest of the paper is organized as follows: Section 2 gives an overview of relevant design advisor issues, Section 3 describes the approach used with the MDC Advisor, Section 4 describes experiments with the MDC Advisor, and we conclude with Section 5.

2. Background

2.1 DB2 UDB V8.1 MDC implementation

In the MDC implementation in DB2 UDB V8.1 proposed by Padmanabhan et al. [15], each unique combination of dimension values forms a logical cell that is physically organized as blocks of pages, where a block is a set of consecutive pages on disk. Every page of the table is part of exactly one block, and all blocks of the table consist of the same number of pages. The clustering dimensions are individually indexed by B+ indexes, known as dimension block indexes, which have dimension values as keys and block identifiers as key data.

The DB2 UDB implementation was chosen by its designers for its ability to co-exist with other database features such as row-based indexes, table constraints, materialized views, high-speed load, and mass delete.

Figure 1 illustrates these concepts. It depicts an MDC table clustered along the dimensions year(orderDate), region and itemId. The figure shows a simple logical cube with only two values for each dimension attribute. Logical cells are represented by sub-cubes in the figure and blocks by shaded oval, and are numbered according to the logical order of allocated blocks in the table. We show only a few blocks of data for a cell identified by the dimension values <1997,Canada,2>.

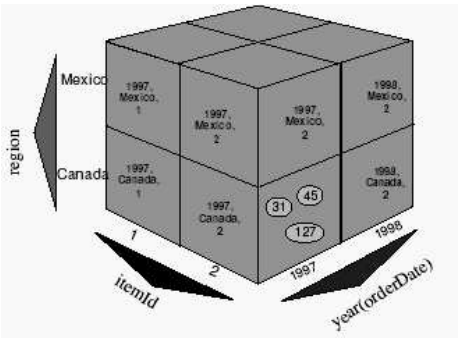

Figure 1: Logical view within an MDC table

2.2 Cost-based evaluation for database advisors

Lohman et al. [13] suggest using cost estimates provided by the database optimizer as part of the evaluation engine of an index advisor that recommends table indexes. In this model, Lohman et al. used a simulation technique to determine access cost impact of potential table indexes. Tthe DBMS is taught to consider "virtual" indexes within its query compiler, resulting in an effective costing of query performance.

The key advance in Lohman's technique is the use of optimizer estimates to evaluate the value of a potential change in the design of a database. The empirical results for this technique were found to be quite good for index selection. A variation of this method was exploited again for another physical database design problem in [18] to design partitioning schemes for shared-nothing massively parallel processing (MPP) databases.

The idea of reusing the database optimizer's cost estimations for evaluating cost benefit of physical design changes in the database is based on the observation that the query optimizer's cost modeling is sensitive to both the logical and physical design of a database. Having the model for workload resource consumption allows us to exploit this model for "what-if" analysis.

In the area of automating MDC dimension selection, there are implementations such as WARLOCK [21], which was limited to parallel warehouses for shared- everything or shared-disk architectures. It used its own cost model instead of using the database engines.

2.3 Estimating the cardinality of distinct values in a set from a data sample

The ability to estimate the cardinality of a set from a sample is an important aspect of MDC design. This topic was surveyed in depth in the 1990s, notably in [9][10].

The known estimators can be divided into two main categories: i) those that evaluate cardinality while examining the frequency data in the sample, and ii) those that generate a result without considering frequency distribution across classes in the sample. The latter type are significant to this paper because they can be calculated easily with only a small set of input variables describing the sample, such as sample frequency, sample size and the cardinality of unique values in the sample. The best of these latter estimators is the First Order Jackknife estimator, which can be described as follows:

When the data set contains no skew the scale-up factor, defined as *Scale = D/E[d]*, is given by

$$Scale = D / E[d] = 1/(1-(1-q)^{(N/d)}) \qquad (1.)$$

Here *D* is the number of distinct values in the set and *d* is the number of distinct values in the sample. Also, *E[d]* is the expected number of distinct values in the sample under Bernoulli sampling with rate *q = n/N*, where *n* is the sample size and *N* is the set size. *E[d]* is the theoretical expected value of *d*, i.e., the average value of *d* over many repeated samples. The idea behind the "method of moments" estimator is to derive an equation relating *E[d]* to *D*, based on theoretical considerations. We solve for *D* to get a relation of the form:

$$D = f(E[d])$$

for some function f. Our estimator $\hat{D}$ is then obtained by substituting *d* for *E[d]* in the above relation:

$$\hat{D} = f(d) \qquad (2.)$$

Such a substitution is reasonable if the sample is not too small. *E[d]* is the "first moment" of *d*, so we are replacing a moment by an observed value.

2.4 Expression based columns

Some popular RDBMS products available today provide the ability to define expression-based columns as part of a relational table definition. These columns, sometimes called generated columns or virtual columns, are mathematical functions of columns within their record tuple. For example, one might define an expression-based column on an employee table that is a function of each employee's SALARY as follows:

```
CREATE TABLE EMPLOYEES
        ( EMPLOYEE_ID INT,
        SALARY DECIMAL(10,4),
        SALARY_RANGE INT
        GENERATED ALWAYS AS
            ( SALARY/1000 ))
```

These expression-based columns based on mathematical and lexical models in many cases have superior clustering potential over one or more base columns. For example, INT(SALARY/1000) is likely to be superior in terms of clustering potential to clustering directly on SALARY.

3. MDC Dimension Selection

MDC requires the allocation of storage blocks to disk for all cells (unique combinations of dimensions) that have at least one tuple. Since in practice all cells will have at least one incompletely filled block, MDC will generally cause some storage expansion. Since storage may be constrained and does impact system performance, it is treated as a constraint on the selection problem. Accordingly, MDC solutions are considered only if they require no more than 10% extra space than a non-MDC implementation. 10% was chosen as a reasonable trade-off to a) constrain increased storage costs and b) constrain any possible negative effect that storage increase may have on queries processing data along access patterns that do not benefit from MDC.

With this constraint in mind, we exploit the SQL query optimizer to model the resource consumption of the workload with and without MDC clustering. Once a set of candidate dimensions,, and their respective benefits to the workload, is identified, we also model how each dimension's benefit will degrade at various coarsifications. Finally, through a search and sample process, the space of possible combinations of dimensions and their coarsifications is examined to find the MDC design for a table that maximizes the combinations of dimensions while satisfying the expansion constraint.

The search space for selecting clustering dimensions is huge. The basic problem of selecting clustering dimensions from a finite set can be modeled easily as a simple combination problem. However, since each dimension has some number of degrees of coarsification, the search space expands exponentially. Assuming an equal number of degrees of coarseness for each dimension, the following equation shows the combinations of "*n*" dimensions each with "*c*" degrees of coarsification:

$$\left(\sum_{r=1}^{n-1}((n!)/(r!(n-r)!))c^r\right) + c^n \qquad (3.)$$

This equation takes a standard formula for the combination of n items, and expands based on the fact that, for each iteration of the sum, each tuple has its

combinations expanded by a factor of c^r because each part of the tuple has c degrees of coarsification (i.e., c ways in which it can be selected). Similarly, the formula concludes with c^n since the selection space for a selection that includes every dimension, each being selected at one of c degrees, is c^n. In general, not all dimensions have the same number of degrees of coarsification. Even so, equation (3) suggests the complexity of the space.

In Subsection 3.1 we give an overview of the methodology adopted, and in subsequent subsections we expand on some key areas. This methodology expects the following inputs from the user:

1. A workload specification, detailing specific queries and the frequency of execution of each.
2. A sample database including the database tables, indexes, and a sample of data. The more complete this database is, the better the recommendations.

Note that the MDC Advisor was designed as an extension to the DB2 Design Advisor, which supports several techniques for automated workload capture, compression, ranking, and weighting [11][17].

3.1 High-level overview of the MDC selection model

Our approach is based on searching over the constellation of combinations of dimensions at various coarsifications to find a combination that has the highest expected benefit while satisfying the storage expansion constraint.

1) Identify candidate clustering dimensions and their maximal potential workload benefit:
   a) Baseline the expected resource consumption (via SQL optimizer estimates) of each query in the workload with all optimizer clustering statistics simulated to represent poor clustering.
   b) Each query in the workload is reoptimized in a special mode, whereby the SQL optimizer simulates the effect of clustering on all candidate clustering dimensions. The dimensions are selected by their use in predicates, as described in Section 3.2. During this phase the optimizer is essentially modeling a best-case scenario where the data is clustered perfectly along all potentially useful clustering dimensions. Also, during this phase we are modeling the maximum potential benefit of MDC apart from its total storage requirement. The clustering dimensions are modeled within the query compiler/optimizer at the finest level of granularity possible for each dimension as if that dimension was the only clustering dimension used. This granularity is titled the Finest Useful Dimensions Granularity (FUDG, pronounced "fudge"), and represents an upper bound on the granularity of each dimension that satisfies the storage expansion constraint. At the FUDG coarsification, a single dimension can be reasonably useful as a clustering dimension while still populating most of the storage blocks. The maximum cardinality of cells is deterministic, as described in Section 3.3, and can be used directly in the optimizer virtual simulation.
   c) Contrasting 1a and 1b we can determine which virtual clustering dimensions in 1b resulted in significant positive differences in the access plans and resource consumption of the queries. The relative reduction in query resource consumption (estimated by the query optimizer) provides an estimate of the benefit gained by clustering on each candidate dimension at its FUDG coarsification.
2) Generate a search space of candidate MDC keys:
   a) A list of candidate dimension and their maximal potential contributions was generated in the previous step. We begin the next phase by designing potential coarsifications of each dimension (where supported): for example, SALARY/1000, SALARY/2000, SALARY/4000, etc., described in detail in Sections 3.4, 3.5, 3.6). A sample of data for each table is then collected. This sample includes a small percentage of tuples from the base but covers exclusively the clustering dimensions identified in step 1c above. The sample data also includes generated expressions that define the coarsifications for each dimension.
   b) Statistics are then collected regarding the cardinality of distinct values for each column in the sampled data, and extrapolated by means of the First Order Jackknife estimator to estimate the cardinality of each dimension at the various degrees of coarsification considered.
   c) The maximum potential clustering benefit or each dimension was determined in step 1c but only at the FUDG coarsification. Now for each dimension its potential value will be estimated again at each of the coarsifications considered, with the assumption that benefit generally decreases as coarsification increases. The benefit attenuation is determined by a curve- fitting process, described in Section 3.7. This yields an expected benefit for each coarsification of each dimension.
   d) For each table, a set of candidate clustering keys is then generated, forming a search space, as per Section 3.8. Each key in the set includes a possible final clustering solution for an individual table. The generated keys are produced by a weighted randomized search. With just one or two candidate dimensions, it is possible to perform an exhaustive search, but

with more dimensions the search space can be prohibitive.

This process yields a set of candidate MDC keys (i.e., potential designs) for each table.

3) For each table, test the candidate MDC clustering keys for satisfaction of the storage expansion constraint.
   a) The candidate clustering keys in 2d are sorted by expected benefit. Benefit is assumed to be the sum of the estimated benefits of the parts (i.e., individual dimensions) for the key. This is not entirely accurate, but a sufficient simplification.
   b) For each table, the candidate clustering keys are then evaluated for space constraint. The space consumption for each candidate key is evaluated through a sampling process (Section 3.9.2) and keys that exceed the space expansion constraint are rejected.
   c) Since the candidate keys were first sorted in rank order (by expected benefit), the first candidate key that satisfies the storage expansion constraint is selected as the winner.
   d) This process is repeated for subsequent tables, until all tables identified in 1c have been evaluated.

Note that phase 1 of this analysis (Select dimension candidates) is done across all tables simultaneously by simulating virtual clustering across all referenced tables in the workload. One of the important observations by Lohman et al. is that early algorithms for index selection assumed separability, such that design decisions on one table could be made independently of design decision for another table (specifically in the case of index advisors). However, Lohman et al. [13] observe that this assumption is not always true, as in the case of a nested loop join between relations *R* and *S* where an index on one of *R* or *S* reduces the need for an index on the other. This is so because as long as one of *R* and *S* has an index, the join predicate can be applied to the inner relation. Thus, it appears that, at least in the case of joins between relations, data access patterns are co-dependent, and design decisions should not be made for each table in complete isolation. The same arguments apply to the problem of MDC design, so separability should not be assumed. Our approach is a hybrid in which we modeled dimension interdependency in steps 1 and 2 above, but assumed independence in 3a.

3.2 Identifying candidate columns

Candidate clustering columns are identified during optimization of SQL queries and the simulation of virtual MDC: these include columns that are used for predicates and operators and are likely to benefit from clustering, such as:

- GROUP BY,
- ORDER BY,
- CUBE,
- ROLLUP,
- WHERE predicates for equality, inequality, and ranges.

3.3 Modeling space waste from table conversion to MDC

Figure 2 illustrates several cells each containing a number of storage blocks, with the final blocks in each cell only partially filled. The greater the number of cells there are, the more partially filled blocks, and therefore the more space wasted. An estimate of the space waste can be made by assuming each cell contains a single partially filled block at the end of its block list. The space waste is then:

$$W = \eta_{cells} \cdot \mathrm{P}_{\%} \cdot \beta \qquad (4.)$$

where $\mathrm{P}_{\%}$ is the average percentage of each storage block left empty per cell, and β is the blocking size. On average, the last block in each cell will be 50% filled, except in cases of largely empty cells (very few tuples in the cell). In the presence of either data skew, or very high cell cardinality, the number of cells with very few tuples may increase, resulting in a high vacancy rate in the final block of some cell. In fact, the choice of $\mathrm{P}_{\%}$ is not critical, provided it is larger than 50%, since the goal is to observe gross expansion of space rather than to estimate space use accurately. In our implementation, we have used a conservative estimate for $\mathrm{P}_{\%}$ of 65%.

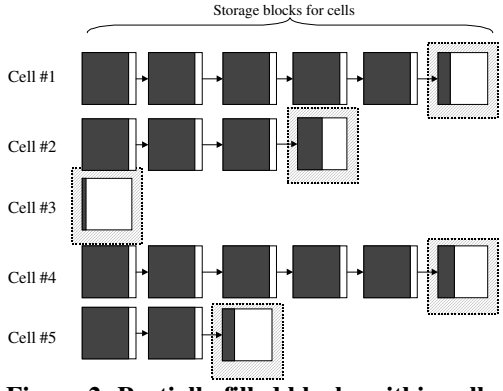

Figure 2: Partially filled blocks within cells

3.4 Coarsification approaches for specific dimension types

For each range dimension, there are specific ways that we can coarsify the clustering, but not an infinite set. In practice, once we have identified FUDG, as illustrated in

the following examples, there are approximately 4 to10 degrees of useful coarsification that we can apply. For example, when coarsifying a date field, we can imagine the following possibilities:

day->month->quarter->year

Similarly, for an INTEGER type, we can coarsify the dimension using division, with a logarithmic scale (i.e., divide by 2, 4, 8, 16, etc).

However, since storage expansion will be proportional to the cardinality of cells in the resulting MDC table, clearly to satisfy the expansion constraint the combinations of dimensions in the final solution must be small enough as per equation (4.). Since the cardinality of cells can only grow as dimensions are taken in combination (e.g., AB will have a cardinality of cells >= A or B individually), therefore, the finest useful granularity that is worth considering in the search space for any single dimension must likewise satisfy this constraint. This granularity is known as the FUDG coarsification, and is described in more detail in the next section. In our selection scheme we begin with the FUDG coarsification, and consider further coarsification of the FUDG coarsification for each clustering dimension showing workload benefit.

3.5 Determining the FUDG coarsification for a candidate clustering dimension

For numeric types, coarsification begins by calculating the FUDG coarsification using the HIGH2KEY statistic (second largest column value) and LOW2KEY statistic (second smallest column value) to define the range of the dimensions, then defining an expression that divides that range into η_{cells_max} ranges (cells). If the base column has cardinality that is below the FUDG cardinality, then the base column defines the FUDG coarsification for that candidate dimension (i.e., this column's FUDG coarsification is simply the base column itself and requires no coarsification).

We define a mathematical function that divides the range between HIGH2KEY and LOW2KEY into a number of ranges, where the number of ranges is the same as the maximum number of cells possible in the table given the space constraint, as shown in Figure 3. HIGH2KEY and LOW2KEY are assumed to represent the reasonable range of values for the dimension.

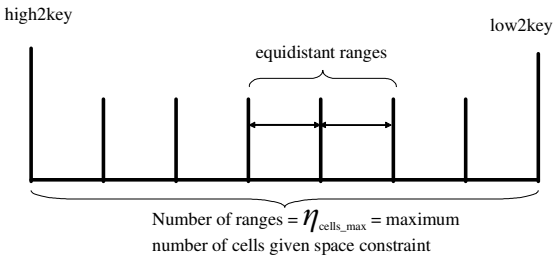

Figure 3: Calculating FUDG for numeric types

$$D_{FUDG} = (Column - LOW2KEY)/iCoarsifier \quad (5.)$$

where *iCoarsifier* is....

$$iCoarsifier = ((HIGH2KEY - LOW2KEY)/iNum\_blocks\_min); \quad (6.)$$

and *iNum_blocks_min* is...

$$iNum\_blocks\_min = MAX(1, table\_size/\beta); \quad (7.)$$

In (7) table_size is the size of the table we are evaluating for MDC, and β is the size of the storage blocks in the cell-block model. In order for this process to work, it is necessary that the dimension be converted to integer form (so that the cardinality of the resulting range is discrete). For real types (DECIMAL, FLOAT, DOUBLE) this means ensuring that they have a substantial positive range. To accomplish this, the FUDG coarsification for Real types includes a multiplicative factor that ensures HIGH2KEY is > 1000.

For DATE and TIMESTAMP fields, we coarsify by casting first to INT and BIGINT, respectively, then using integer division to coarsify to week, month, quarter, year. Special assumptions are made when determining the FUDG coarsification for DATE and TIMESTAMP because of the practical concern that the data currently in the database at the time the MDC Advisor is run may only be a time-fragment of the real data (for example one month's worth of data). This is a very significant and realistic situation for database designers. If only a single month of data were provided, the cardinality of cells in the DATE dimension might be limited to 31 distinct values, while in fact the data may only be a one month sample of a seven-year data warehouse. Therefore, to mitigate this risk, in both TIMESTAMP and DATE cases, we assume that WEEK of YEAR is a reasonable estimate of FUDG since it coarsifies the column to a maximum of 52 cells per year. We do not recommend clustering on DATE or TIMESTAMP without coarsification, even when the apparent cardinality of cells in the dimension data is low enough.

3.6 Sampling for cardinality estimates

For each dimension, once the FUDG coarsification has been estimated, further coarsification is designed. The search model requires a reasonable estimate of the cardinality of each dimension at each of the dimension coarsifications (during step 2b of the process described in 3.1 above), as well as the ability to measure the cardinality of combinations of these dimensions (during step 3a in Section 3.1 above). To facilitate a reasonable response time for the MDC Advisor, a sampling approach is used. In this sampling model, data is sampled for each candidate table only once, and stored in a temporary staging table. The sampling is performed using a Bernoulli sampling method. Statistics including cardinality of dimensions, dimension coarsifications and combination of dimensions can be collected over the sampled data rather than the base table, which enables the evaluation of a large number of variants while only sampling the base table once. While the staging table holding the sample may need to be scanned multiple times, significant performance benefit accrues from the fact that the staging table is a small fraction of the size of the base table from which its data came.

Cardinality estimation research [9][10] suggests that the accuracy of statistical cardinality estimators drop off precipitously when sampling rates fall below 1%. Therefore, the staging table constructed here uses the larger of a 1% sample or a sample of 10000 tuples to construct its sample.

The staging table T_{temp}, includes a definition of all the base columns from T_{base} that are candidate clustering dimensions. In addition, T_{temp} includes expression-based columns for all of the coarsification of the base columns the MDC Advisor will consider, starting with the FUDG coarsification level, and increasing from there. For example, if SALARY may have a FUDG coarsification of $SALARY_f$ = SALARY/1000, we may also create generated columns of $SALARY_f$ /4, $SALARY_f$ /16, $SALARY_f$ /64..., etc. The staging table is populated with a 1% sample from the base table. This allows the cardinalities of unique values that are needed in 2c and 3a of Section 3.1 to be counted while only taking the sample once (i.e., sample once, count many).

3.7 Modeling workload benefit consequences of clustering coarsification

One of the key issues is to understand the likely effect of coarsification on the expected benefit in clustering on any given dimension. A brute force approach to solving this problem would be to re-evaluate (simulate) the workload cost with each individual coarsification of each dimension, or perhaps all possible combinations. Such an approach for workloads of any significant dimensionality is impractical. Instead we use a simple model sufficient for the MDC selection process, based on the following two observations

1. When a database table has only one cell, MDC provides no value.
2. Expected benefit at the FUDG coarsification was determined through simulation within the SQL optimizer.

This gives us two points of reference on a performance versus cardinality of distinct values graph, when cardinality is 1 (i.e., zero benefit) and at the cardinality of distinct values at the FUDG coarsification. We also infer that the benefit due to clustering is monotonic and decreasing as coarsification increases.

Although the exact shape of the monotonic curve cannot be easily determined, we have modeled it as a smooth logarithmic relationship, such that the penalty for coarsifying a dimension is initially minor, but increases dramatically at higher coarsification levels. We apply a curve-fitting process to plot a concave polynomial between the two well-known points to derive a benefit-coarsification function, as per **Figure 4**. From this relationship function, we can model the performance benefit of any coarsification level of a dimension given its cardinality of cells at the FUDG coarsification level.

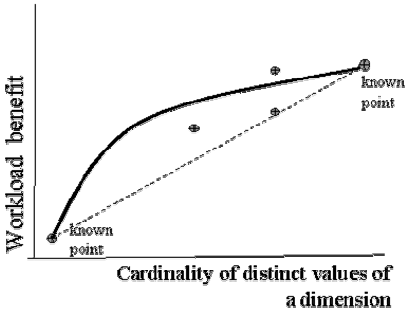

Figure 4: Curve-fitted benefit-coarsification function

The benefit versus cardinality of cells function is then determined as follows in equations (8) and (9).

$$B = m * \log(C) \quad (8.)$$

$$m = Bf/(\log(Cf)) \quad (9.)$$

B is the performance benefit at a given coarsification level, and ***C*** is the cardinality of cells at the same coarsification level. ***Bf*** is the performance benefit at the FUDG coarsification and ***Cf*** is the cardinality of cells at the FUDG coarsification level for the dimension.

3.8 Search algorithm

To find an optimal combination of dimensions that satisfies our storage expansion constraint, we have used a simple weighted randomized search, which includes some qualities of a genetic algorithm including weighted selection of attributes. After completing steps 1a, 1b, and 1c described in Section 3.1, the algorithm is left with a search problem that must select the best possible MDC design given a list of candidate dimensions with estimated performance benefit at their FUDG coarsification. The search space for this problem is all possible combinations and permutations of all these candidate dimensions at any of their possible coarsifications. The complexity of this search was described in equation (3). Since the evaluation of a candidate clustering key requires some degree of cardinality evaluation (to ensure the storage constraint is not exceeded) the evaluation function requires some sampling and counting. Therefore, the need for cardinality estimation requires a computationally costly evaluation function, and an exhaustive search is not practical.

Using our weighted randomized search, combinations of dimensions at various coarsifications are selected in probabilistic proportion to their relative benefit to the workload. Each such combination forms a candidate solution. The set of generated candidate MDC keys (i.e., solutions) are then sorted by benefit. For simplicity, the benefit of each MDC solution is assumed to be the sum of the workload benefit for each dimension in the solution. Once the candidate clustering keys have been generated and ranked, they are evaluated in rank order using the evaluation function described in the previous section to determine whether they satisfy the storage expansion constraint. Since the candidate clustering keys are sorted in rank order, the first candidate key to pass the test for storage expansion is chosen as the final clustering recommendation for a given table.

To improve the efficiency of the search, when a candidate key is found to have a design that will lead to gross storage expansion (e.g., >5x storage growth), then we reject this key, and also eliminate near neighbours in the search constellation. This near-neighbour reduction has been effective in high dimensionality search spaces in greatly reducing the search cost. On our experiments, the efficiency of the search was improved by 400% in some cases by this addition.

3.9 Evaluation function for candidate keys

3.9.1 Estimating workload benefit

The search method used in this paper will require an evaluation function to assess the fitness (or value) of each search point in the candidate solution space. The "value" in this context is the potential benefit to the query workload in improving performance. To do this we exploit a variation of the technique used by Lohman et al. [13] where the database optimizer is used to provide a cost estimate of the workload. In this method the optimizer is given a simulation of the table definition and table statistics (and statistics for dependent objects) against which it makes its estimations. In the case of MDC the problem is more complex for these reasons:

- MDC affects the base table: it is not simply an optional attachment, as in the case of an index;
- MDC affects the statistics of the base table, namely table size;
- The MDC search typically includes search points for dimensions at multiple degrees of coarsification.

To deal with the complexities, the optimizer model is extended to model MDC candidates, affecting statistics of the base table as well as cluster ratios on existing indexes. The benefit of the FUDG coarsification of a dimension is then calculated as the aggregate of the resource consumption reduction for each query in the workload that exploits the virtual clustering dimension as compared to the same resource consumption analysis without MDC clustering. However, the cardinality of cells at a given coarsification of a dimension cannot be reliably estimated, and sampling is required to determine this. Once a reasonable estimate of the cardinality of cells is obtained, the attenuated workload benefit due to coarsification of a dimension can be estimated using equation (8) as described in Section 3.7. Therefore, the SQL query optimizer is used to estimate the workload benefit for each dimension at its FUDG coarsification, while sampling, counting, and curve fitting are used to estimate the benefit of the same dimension at increased levels of coarsification.

Once the benefit of each candidate dimension is calculated at its FUDG coarsification, the expected benefit for each dimension at further coarsifications is modeled through the process described in Section 3.7 and the curve-fitting algorithm described there, provided cardinalities or estimates of cardinalities are known for each coarsification of the dimensions we wish to model. These estimates of cardinality for each candidate dimension are similarly detected through the sampling process described in Section 3.6, and extrapolated using the First Order Jackknife Estimator.

Using these methods in combination, we now have a model for:

a) Detecting candidate dimensions.
b) Estimating the workload benefit of a candidate clustering dimension at its FUDG coarsification.

c) Modeling the benefit of each candidate clustering dimension at coarsifications beyond the FUDG coarsification, as a logarithmic function of cardinality reduction.

3.9.2 Evaluating satisfaction of the storage expansion constraint

The remaining problem in the evaluation function is to determine for any given combination of dimensions and coarsifications what the cardinality of resulting cells will be. This measure of cardinality of cells is critical to determine in order to satisfy the storage expansion constraint. Using the same data sample collected above into $T_{temp,}$ we can use SQL to count the cardinality of the unique values of a combination of dimensions that correspond to the dimensions in a MDC solution we being evaluated.

To do this, we use an SQL query such as the following:

SELECT COUNT(*) FROM (SELECT DISTINCT A,B,C FROM T-TEMP)
AS CELL_CARD; (10.)

This returns the COUNT of distinct values of the clustering key (ABC). Once the cardinality in T_{temp} of distinct values of the candidate clustering key is determined, we can scale this sampled cardinality using the First Order Jackknife Estimator to estimate the number of cells that would exist in the entire table. This sampling and extrapolation method effectively models correlation between the dimensions in a candidate solution.

Once the cardinality of cells is estimated, it can be tested against equation (4) to determine whether the storage expansion constraint is satisfied.

3.10 Data skew

In a few instances (see 3.6, 3.7 and 3.9.2), the MDC Advisor algorithm requires a statistical estimator to extrapolate the cardinality of unique values in a sample. The First Order Jackknife Estimator was chosen for its simplicity. This estimator is known to be weak in the presence of severe data skew. Though length limitations do not allow for a detailed analysis here, it can be shown that the specific requirements in this algorithm are quite tolerant to estimation inaccuracies, which allow the First Order Jackknife estimator to be adequate in the presence of data skew in most cases. Even so, several other estimators with superior skew handling are described in [9][10], which can be substituted to improve the robustness of the algorithm.

4. Experimental Results

4.1 Test Objectives & Description

The objective of the tests was to compare the quality of the MDC Advisor recommendation when compared to expert human recommendation against a well-known schema and workload. The industry standard TPC-H benchmark was used for the tests [1]. The metric used for comparison is called the TPC-H Composite Query-per-Hour (QphH@Size). For the experiments a 10 GB TPC-H database running on DB2 UDB V8.1 on a pSeries® server with AIX® 5.1, 4 X 375 MHz CPUs and 8 GB RAM was used. Six experimental tests were performed:

1. Baseline: The performance of the benchmark without MDC. Table 1 describes those tradition RID (row) indexes used for the baseline experiment, which had cluster ratio quality of 5% or better, a measure of percentage of data that is well clustered along one dimension.
2. Advisor 1: The performance of the benchmark using the top most MDC design (described in Table 2) of the Advisor.
3. Advisor 2: The performance of the benchmark using the second best MDC design (described in Table 3) for the Advisor.
4. Expert 1: The MDC design used during IBM's most recent 2003 TPC-H publication. This is described in Table 4. According to TPC-H guidelines, the MDC design was constrained to clustering exclusively on base columns (coarsification was not permitted).
5. Expert 2: The top MDC design provided by the DB2 MDC development team described in Table 5.
6. Expert 3: An alternative MDC design provided by the DB2 MDC development team is described in Table 6.

Index name	Base table	Columns (key parts)	**Cluster quality (%)**
L_OK	LINEITEM	+L_ORDERKEY	**100**
R_RK	REGION	+R_REGIONKEY	**100**
S_NK	SUPPLIER	+S_NATIONKEY	**36.8**
PS_PK_SK	PARTSUPP	+PS_PARTKEY +PS_SUPPKEY	**100**
S_SK	SUPPLIER	+S_SUPPKEY	**100**
PS_PK	PARTSUPP	+PS_PARTKEY	**100**

Table 1: Single dimensional clustering in baseline

Base table	MDC dimensions
CUSTOMER	C_NATIONKEY,C_MKTSEGMENT
LINEITEM	(INT(L_SHIPDATE))/7, L_RETURNFLAG, (INT(L_RECEIPTDATE))/14, L_SHIPINSTRUCT
ORDERS	(INT(O_ORDERDATE))/7, O_ORDERSTATUS
PART	P_SIZE
PARTSUPP	(((PS_PARTKEY)/(((1999999 – 2)/(19956))*(8))))
SUPPLIER	S_NATIONKEY

Table 2: MDC design for "Advisor 1"

Base table	MDC dimensions
CUSTOMER	C_NATIONKEY/2, C_MKTSEGMENT
LINEITEM	(INT(L_SHIPDATE))/14, L_RETURNFLAG, (INT(L_RECEIPTDATE))/7, L_SHIPINSTRUCT
ORDERS	(INT(O_ORDERDATE))/14, O_ORDERSTATUS
PART	P_SIZE/2, P_CONTAINER
PARTSUPP	(((PS_PARTKEY)/(((1999999 – 2)/(19956))*(16))))
SUPPLIER	S_NATIONKEY/2

Table 3: MDC design for "Advisor 2"

Base table	MDC dimensions
LINEITEM	L_SHIPDATE
ORDERS	O_ORDERDATE

Table 4: MDC design for "Expert 1"

Base table	MDC dimensions
CUSTOMER	C_NATIONKEY
LINEITEM	(INT(L_SHIPDATE))/100, L_SHIPMODE, L_SHIPINSTRUCT
ORDERS	O_ORDERDATE
SUPPLIER	S_NATIONKEY

Table 5: MDC design for "Expert 2"

Base table	MDC dimensions
CUSTOMER	C_NATIONKEY,C_MKTSEGMENT
LINEITEM	(INT(L_SHIPDATE))/100, L_SHIPMODE, L_SHIPINSTRUCT, (INT(L_RECEIPTDATE)/10000
PART	P_SIZE, P_BRAND

Table 6: MDC design for "Expert 3"

The TPC-H workload was run three times for each test; the shortest run for each design is noted here. Execution time variability was found to be quite minimal among the three runs, generally less than 2%. The tests were done with identical database and database manager parameters.

4.2 MDC Advisor search space

A graphical display of search points considered by the MDC Advisor algorithm (Figure 14) for the two largest tables, LINEITEM and ORDERS illustrates some interesting search characteristics. The shaded areas covering the rightmost portions of the space are areas where the search points would have caused severe table storage expansion. As a result, these high expansion candidates are not practical as solutions and are simply rejected from the candidate solution set.

Figure 5 shows the performance benefit versus storage expansion projected for each candidate solution explored in the MDC search. Note that the benefit model assumed < 10% growth, so that candidate solutions resulting in more than 10% growth have bogus benefit. The density of search points that lie along a region in the x domain between 1.0x and 1.1x expansion is quite reasonable, illustrating that the search algorithm is successful in finding many candidate solutions in the acceptable range of expansion. The circled area shows the keys with highest benefit and reasonable data expansion from which the final recommended MDC solution is chosen.

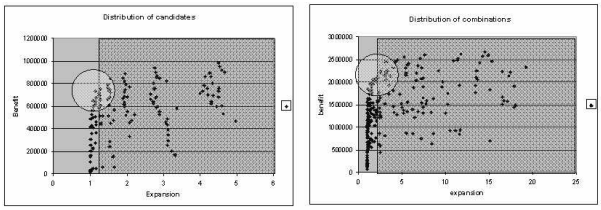

Figure 5: Distribution of search points for TPC-H two largest tables

4.3 MDC table expansion

Table 7 shows the actual table expansion rates for the TPC-H tables for the six clustering designs.

The MDC Advisor logic, was quite effective at selecting MDC designs that were constrained to the space constraint goal of 10% expansion. The largest table expansion was seen in Advisor 1 experiment where LINEITEM table expanded by 11.98%, and 12.76% expansion on PARTSUPP, which is quite good given the 1% sampling rate of the First Order Jackknife estimator.

Table name	No MDC	Expert 1	Expert 2	Expert 3	Advisor 1	Advisor 2
	Size (4K)	Growth (%)	Growth (%)	Growth (%)	Growth (%)	Growth (%)
LINEITEM	2081040	1.05	4.69	8.42	11.98	11.95
ORDERS	443840	4.08	4.08	0.00	5.23	4.89
PART	76240	0.00	0.63	5.56	0.63	9.99
PARTSUPP	319296	0.00	0.00	0.00	12.76	6.49
CUSTOMER	69168	0.00	0.35	1.50	1.50	3.63
SUPPLIER	4096	0.00	7.81	0.00	7.81	6.25
Total	**2993680**	**1.34**	**3.90**	**6.03**	**10.53**	**10.07**

Table 7: Table expansion with MDC

Also the expert designs by human designers (Expert 1, Expert 2, and Expert 3) were generally more aggressive than the MDC Advisor in constraining space expansion (1.34%, 3.90% and 6.03% total expansion, respectively), a likely reflection of their deep knowledge and many years of experience with the TPC-H workload

4.4 Query performance results

The MDC Advisor completed its design analysis and reported its recommendations in less than an hour.

Figure 6 shows the QphH results for the six clustering designs and they show the performance benefit of MDC and the effectiveness of the MDC Advisor algorithm in selecting MDC designs in comparison to human experts.

In these experiments, all of the MDC designs showed significant benefit over the baseline throughput. The rank ordering of the five MDC designs according to their performance benefit Advisor 2 with 11.12%, Expert 1 with 13.35%, Expert 3 with 14.20%, Advisor 1 with 14.54%, and Expert 2 with 18.08%. Significantly, Advisor 1, which represents the MDC Advisor's best recommendation was measurably superior to to MDC Advisor 2, and both Expert 1 and Expert 3.

Also revealing is a view of the performance by individual queries, as shown in Figure 7. No single clustering design achieved gains across the entire workload, highlighting the complexity of the search problem. Specifically, a successful advisor algorithm must consider the overall benefit of clustering designs across all tables and all queries, which is one of the highlights of the approach described in this paper.

5. Conclusion and future work

5.1 Summary

The MDC Advisor algorithm leverages past work in automated physical database design and statistical modeling, in combination with new ideas on MDC, to provide a method for automating the design problem of MDC. To our knowledge, this is the first published algorithm to tackle this important problem. The algorithm exploits a combination of query optimizer what-if analysis, weighted randomized search, data sampling, and statistical extrapolation. Six experiments were performed using a 10 GB TPC-H database to compare the advisor designs against those of human experts and it was found to provide design recommendations that were in line with the quality of these experts. The advisor was effective at modeling correlation between dimensions through sampling, and was able to limit the database expansion under MDC to a value very close to its design goal of 10%. Based on the value shown through these experiments and the importance of the studied problem, the model described in this paper has been implemented for the V8.2 release of DB2 UDB for Linux, UNIX and Windows.

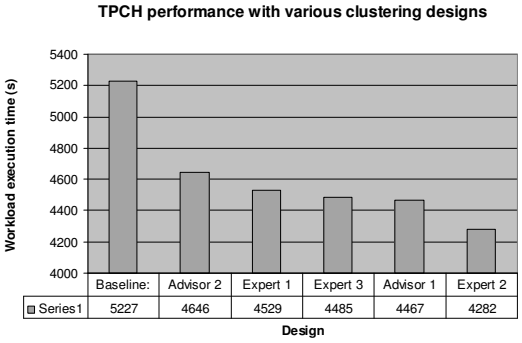

Figure 6: TPC-H overall results

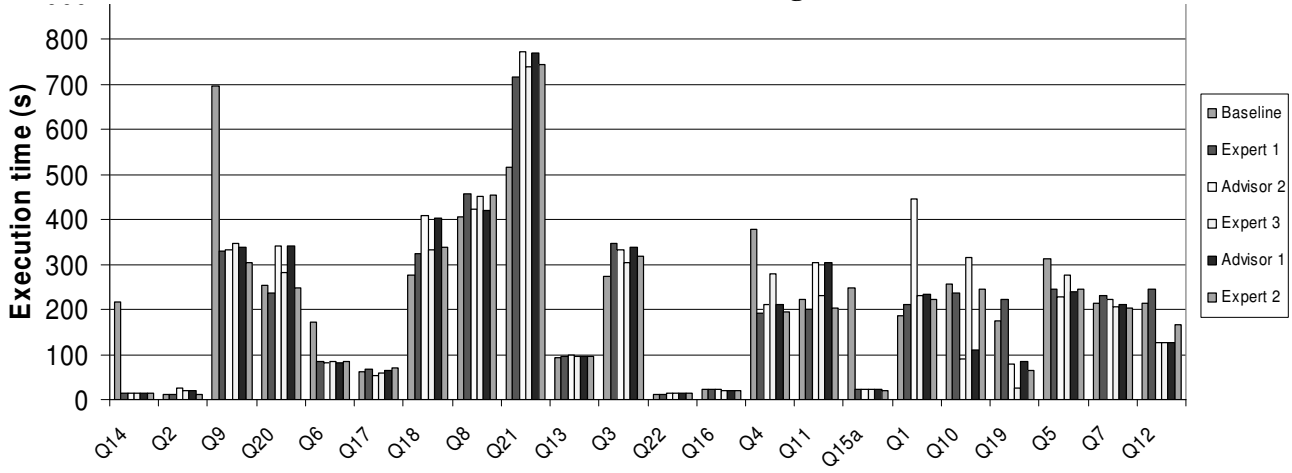

Figure 7: TPC-H query performance for all 6 experiments

5.2 Future work

This work can be enhanced by investigating:

- Hierarchy climbing for dimension coarsification.
- Improving coarsification models and storage estimates in the presence of data skew.
- Efficient table migration (alter) schemes for conversion to MDC.
- Recommendation of block size and adaptive blocking sizes, to better accommodate data skew.
- Improved selection of the storage expansion constraint, including adaptive algorithms.
- Experimentation on larger/varied data sets, schemas, and workloads, in particular including database schemas and workloads from user environments

References

[1] "Transaction Processing Performance Council" http://www.tpc.org/default.asp.

[2] "DB2 Universal Database for Linux, UNIX and Windows" http://www-306.ibm.com/software/data/db2/udb/

[3] S. Agrawal, S. Chaudhuri, V.R. Narasayya, "Automated selection of materialized views and indexes in SQL databases". Proc. VLDB 2000, Cairo, Egypt

[4] B. Bhattacharjee, S. Padmanabhan, T. Malkemus, T. Lai, L. Cranston, M. Huras, "Efficient Query Processing for Multi-Dimensionally Clustered Tables in DB2", Proc. VLDB 2003, Berlin, Germany

[5] S. Chaudhuri, E. Christensen, G. Graefe, V. Narasayya, M. Zwilling, "Self-Tuning Technology in Microsoft SQL Server", IEEE Data Eng. Bul. 22(2), June 1999

[6] S. Chaudhuri, V. Narasayya, "AutoAdmin 'What-if' Index Analysis Utility", Proc. SIGMOD, 1998, Seattle, USA

[7] S. Chaudhuri, V. Narasayya, "Microsoft Index Tuning Wizard for SQL Server 7.0", Proc. SIGMOD, 1998, Seattle, USA

[8] M.R. Frank, E.R. Omiecinski, S.B. Navathe, "Adaptive and Automated Index Selection in RDBMS", Proc. EDBT 1992, Vienna, Austria

[9] P.J. Haas, J.F. Naughton, S. Seshadri, L. Stokes, "Sampling Based Estimation of the Number of Distinct Values of an Attribute", Proc. VLDB 1995, Zurich, Switzerland

[10] P.J. Haas, L. Stokes, "Estimating the number of classes in a finite population", JASA, V. 93, Dec, 1998

[11] S. Lightstone. D. Zilio, C. Zuzarte, G. Lohman, J. Rao, K. Cheung, "DB2 Design Advisor: More than just index selection", IDUG 2004, Orlando, USA.

[12] J.H Liou, S.B. Yao, "Multi-dimensional clustering for database organizations". Information Systems, 2:187--198, 1977.

[13] G. Lohman, G. Valentin, D. Zilio, M. Zuliani, A. Skelly, "DB2 Advisor: An optimizer smart enough to recommend its own indexes", Proc. ICDE 2000, San Diego, USA

[14] V. Markl, F. Ramsak, R. Bayer, "Improving OLAP Performance by Multi-dimensional Hierarchical Clustering", Proc. IDEAS'99, Montreal, Canada

[15] S. Padmanabhan, B. Bhattacharjee, T. Malkemus, L. Cranston, M. Huras, "Multi-Dimensional Clustering: A New Data Layout Scheme in DB2." SIGMOD 2003, San Diego, USA

[16] N. Pendse, R. Creeth, "The OLAP Report", http://www.olapreport.com/.

[17] J. Rao, S. Lightstone , G. Lohman, D. Zilio , A. Storm, C. Garcia-Arellano, S. Fadden "DB2 Design Advisor: integrated automated physical database design", Proc. VLDB 2004, Toronto, Canada.

[18] J. Rao, C. Zhang, N. Megiddo, G. Lohman, "Automating physical database design in a parallel database.", Proc. SIGMOD 2002, Madison, USA

[19] B. Schiefer, G. Valentin, "DB2 Universal Database Performance Tuning", IEEE Data Eng. Bul 22(2), June 1999

[20] T. Stöhr, H. Märtens, E. Rahm, "Multi-Dimensional Database Allocation for Parallel Data Warehouses", Proc. VLDB 2000, Cairo, Egypt

[21] T. Stohr, E. Rahm, "WARLOCK : A Data Allocation Tool for Parallel Warehouses", Proc. VLDB 2001, Rome, Italy (Software Demonstration)

Trademarks