

Performance Evaluation of Multidimensional Array Storage Techniques in Databases

Norbert Widmann¹, Peter Baumann

Bavarian Research Centre for Knowledge-Based Systems (FORWISS)

Orleansstr. 34, D-81667 Munich, Germany

{widmann, baumann}@forwiss.de

Abstract

Storing multidimensional data in databases is an important topic both in the industrial and scientific database communities. Arrays are offered as a multidimensional data structure by most programming languages. Conventional database systems, however, do not support arrays of arbitrary dimensionality and base type. RasDaMan is a DBMS integrating arrays as a first class data type offering both a declarative query language and a specialised storage structure for arrays.

The work presented here evaluates the performance of queries on multidimensional array data stored in RasDaMan versus storage in a conventional RDBMS. In the relational system, the data is both mapped to relations and stored directly as binary data in BLOBs. The queries executed were modelled after queries common in scientific applications and decision support.

1. Introduction

Multidimensional data is getting more and more relevant in the database community. On the design level, multidimensional data models have been developed [1] enabling the user to present problems from his application domain directly in their multidimensional form. The storage of data, however, is commonly done in RDBMSs mapping the multidimensional model to relations.

Multidimensional arrays are the data type for processing multidimensional data in most programming languages. Support for multidimensional arrays in a DBMS enables users to store data from their application area in a database without mapping it to another data model. The RasDaMan DBMS supports array storage with a declarative query language and specialised storage structures. The query

language offers a number of operations on arrays. RasDaMan was developed at the Bavarian Research Centre for Knowledge-Based Systems (FORWISS) during the ESPRIT Long-Term Research project of the same name.

Performance is a key factor when dealing with array data, as typically large amounts of data are stored and processed. Online archives for remote sensing data are planned with a size of terabytes; data warehouses used as basis for OLAP applications storing hundreds of gigabytes are not uncommon. RasDaMan, as a specialised DBMS, offers an optimising query language and specialised storage techniques specifically geared towards requirements set by applications dealing with multidimensional data [3].

This paper compares the performance of a specialised DBMS with a commercial relational system. Section 2 introduces the RasDaMan DBMS. The storage alternatives for arrays in RDBMSs are discussed in Section 3. The queries and data used for performance evaluation are explained in Section 4 followed by a discussion of the results in Section 5. Section 6 presents our conclusions.

2. The RasDaMan System

The RasDaMan system offers full DBMS support for multidimensional arrays. It is a client/server system providing the declarative query language RasQL and a C++ API called RasLib for users to implement their applications. The logical view on arrays is independent from their physical storage, which is done in tiles [2]. The system is implemented on top of an ODBMS. Query processing, optimisation, and operation execution is all done completely in the RasDaMan server, while the base DBMS is used only for storage. The following gives a short overview of the RasQL query language. For a more complete treatment please refer to publications focusing on implementation of the RasDaMan system [4].

The RasDaMan Query Language, RasQL, extends SQL-92 with operations on multidimensional arrays or parts

¹sponsored by the European Commission in the ESPRIT Domain 4: Long-Term Research under grant no. 20073.

thereof. This is a query from processing remote sensing images: “Return the lower left corner with intensity increased by 10 of all those images in collection *earth* where the intensity of every pixel in an area around the centre of the image is greater than 254.”

```
SELECT img[0:254,0:254] + 10
FROM   earth AS img
WHERE  all_cell (
      img[350:449,350:449] >= 254 )
```

One easily visible difference between SQL and RasQL is access to parts of arrays. In the SELECT clause, a subselect is done on a 2-D array. Other possible operations are reducing the dimensionality by fixing co-ordinates in one or more dimension or accessing the whole range in a dimension using an asterisk as a wild card for one or both dimensional borders. Another class of operations enables modifying cell values of arrays. The addition operation with a constant in this query increments every cell in the selected area by 10.

In the WHERE clause, the expression `img[350:449,350:449] >= 254` compares every cell in the relevant area to 254. The result is a Boolean array containing TRUE for every cell which has a value greater than or equal to 254, FALSE for every other cell. `all_cell` aggregates this Boolean array to one Boolean value: TRUE if all cells are TRUE, FALSE otherwise. The value returned by `all_cell` is used as a condition in the WHERE clause.

3. Storing Arrays in RDBMSs

The relational data model was designed mainly for modelling business data. In the following, two commonly used techniques for storing multidimensional arrays using relational DBMS technology as available in commercial systems on the market are discussed.

3.1 Binary Large Objects

A BLOB is basically a 1-D array of bytes. Multidimensional arrays can be stored in BLOBs, if their content is linearised and complex base types are mapped to bytes. Data interpretation is completely up to the application program; basic operations like conversion between low and big endian and interpretation of data as arrays have to be implemented by the application programmer. Standard SQL does not support operations on BLOBs, they can just be selected as a whole; no indexes can be built on their content. Selection of BLOBs depending on their content can only be done after transferring the whole data to be searched to the client. Each RDBMS vendor provides a proprietary interface to access BLOB data in their API; subselects are only possible on 1-D intervals.

While gaining some of the advantages of DBMS technology like access control and data integrity, BLOBs are still a solution at a semantically low level. The DBMS has only minimal knowledge of their content, and their storage is not optimised towards requirements set by multidimensional data.

3.2 Arrays in Relations

Multidimensional data models can be mapped to relations: For every cell its co-ordinates are materialised together with its value. If more than one multidimensional object is stored in a relation, then additionally the id of the object has to be stored for each cell. Assuming two bytes for each co-ordinate of a multidimensional array, a 3-D array is stored with an overhead of at least six bytes per cell. For efficient access to the cells additional overhead is needed for indexes on the co-ordinates. The results of queries are still relations after transmission to the client, mapping to arrays has to be done in the application.

4. The Test Environment

Our tests were executed on a Sun Ultra I/140 with 256MB of main memory running Solaris 2.5. All data was stored on one local 4GB disk. The RasDaMan server was used together with O₂ Version 5.0.2. The RDBMS used was one of the major commercial systems. A current version was installed according to the instructions in the manual with no special tuning efforts. Both DBMS client and server were executed on one machine for all tests.

4.1 Test Data

Two data sets were used in the tests: A single 3-D volume tomogram from the medical application area (*tomo*) and a set of 2-D images from the remote sensing area (*earth*). The base type in both cases was a 8bit unsigned integer. The raw size of the data sets is as follows: $256 \times 256 \times 154 \text{ bytes} \approx 9.6\text{MB}$ for *tomo* and $15 \text{ images} @ 800 \times 800 \text{ bytes} \approx 9.2\text{MB}$ for *earth*.

Storage overhead using RasDaMan or BLOB based storage is negligible. In the RDBMS the raw data for *tomo* had a size of 76.8 MB and for *earth* 73.6 MB, including the storage overhead for co-ordinates. Indexes were built on each co-ordinate and on the concatenation of all co-ordinates. Storage of sparse data was also evaluated in the RDBMS. Two degrees of density were evaluated on the *tomo* dataset: 10% in relation *tomod10* and 2% in relation *tomod2*. This data was artificially generated by just storing 10% resp. 2% of the cells in the relation. The cells stored were chosen randomly and evenly distributed. The dense dataset will be referenced as *tomod100*.

Using BLOB storage, the tomogram was stored as a single column relation with exactly one tuple containing the whole tomogram data. The earth images were stored in a relation with two columns: a two byte integer with an ID and a BLOB containing the array. These two relations are referenced as *tomoblob* and *earthblob*.

RasDaMan supports different physical storage layouts in the form of tiling. In our tests simple regular tiling schemes were used. The tiles were of quadratic shape and had a size of 64kB.

4.2 Test Queries

Test queries were designed to be close to application areas where multidimensional arrays are frequently used. Three basic query patterns were implemented:

1. A subselect on the 3-D volume tomogram. Subselects are a common operation in all application areas.
2. An operation summing up seven consecutive 2-D planes of the 3-D volume tomogram resulting in one 2-D plane. This operation was modelled after roll-up operations common in OLAP applications, eg summing up weekly sales of products in areas.
3. Content based selection of earth images. In geographic or medical applications, it is common to further analyse only “interesting” images defined by criteria based on the content like, eg, no cloud cover.

RasDaMan

A subselect on a 3-D array in RasDaMan is expressed in the following RasQL query:

```
SELECT img[20:138,10:128,30:101]
FROM tomo_cubed_64 AS img
```

This query selects a query box sized proportional to the shape of the volume tomogram with a selectivity of 10%, ie retrieval of 10% of all cells. Selectivities tested in queries were 0.5%, 1%, 2%, 5%, 10%, 20%, 50% and 100%. The measurements is named *tmov_64*.

The roll-up query is expressed as seven operations selecting a 2-D slice out of a 3-D array combined with addition operations. The result is a 2-D array containing the sums. The following query is an example for a roll-up along the second dimension:

```
SELECT img[:,47,:]+img[:,48,:]+
img[:,49,:]+img[:,50,:]+
img[:,51,:]+img[:,52,:]+
img[:,53,:]
FROM tomo_cubed_64 AS img
```

The asterisk in the spatial dimension denotes the lowest resp. highest index in the corresponding dimension. So along the x and z dimensions the whole data is selected. The seven consecutive slices for the roll-up were chosen randomly for repeated queries.

Content based selection was used as an example for introducing RasQL in Section 2. All images in *earth* were rescaled to grey values from 0 to 249. Three images were modified to contain the same value in all cells: 253, 254 resp. 255. Another three images were modified to contain 250, 251 resp. 252 in exactly one cell. The other nine images were stored without further modification. After this processing it was possible to select one, two or three images by adapting the WHERE clause. In the SELECT clause a subselect of an area of interest is done. This query selects two images using *some_cell*:

```
SELECT img[0:254,0:254]
FROM earth as img
WHERE some_cell (
    img = 250 OR img = 251 )
```

Relations

If array data is stored in relations, it is possible to query it using standard SQL-92 queries. The subselection query can be easily mapped from RasQL to SQL:

```
SELECT x, y, z, val
FROM tomo
WHERE x BETWEEN 20 AND 138 AND
      y BETWEEN 10 AND 128 AND
      z BETWEEN 30 AND 101
```

While the subselect query in SQL is longer than the corresponding RasQL query, the roll-up query can be expressed shorter:

```
SELECT x, z, SUM(val)
FROM tomo
WHERE y BETWEEN 47 AND 53
GROUP BY x, z
```

Transforming content based selection queries to SQL is more difficult. The query selecting two images with *some_cell* was executed using this SQL query (the translation of the *all_cell* query even involves three select statements):

```
SELECT *
FROM earth e1
WHERE e1.x <= 254 AND e1.y <= 254 AND
      e1.id IN (
    SELECT e2.id
    FROM earth e2
    WHERE e2.val = 250 OR e2.val = 251
);
```

BLOBs

To execute the queries on arrays stored in BLOBs, embedded SQL programs had to be written. A program of about 50 lines executes the subselect query by mapping the 3-D query box to a set of 1-D intervals in the BLOB. The roll-up query was not implemented for BLOB based storage, as it would have involved a substantial coding effort. The content based selection transfers the whole data to the client and checks the condition on the client. The code has about 70 lines and all parameters are hard coded.

In general working with BLOBs involves a large programming effort compared to RasQL and the RasDaMan C++ API. Furthermore, queries are hard coded into the application, making it impossible to support interactive explorative querying of data. Changes in application requirements would involve substantial modifications to the code.

5. Performance Results

Benchmarking of the queries was done with almost no additional load on the system. The measurements were repeated, times reported are average times. Standard deviations were very small.

5.1 Subselect Queries

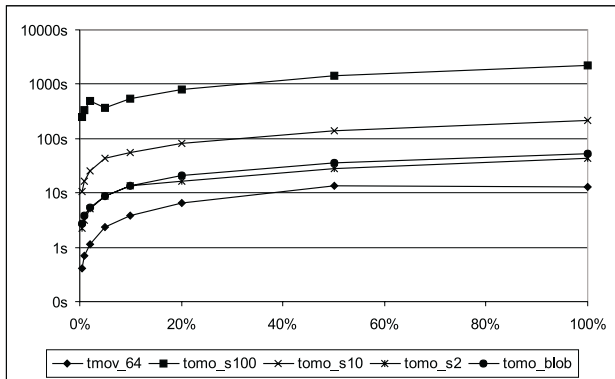


Figure 1. Subselect.

Performance for subselects using the different storage techniques is shown in Figure 1. Storing the dense array in relations, it takes more than 35 minutes to access it as a whole. Performance for all selectivities on *tomod100* is at least one order of magnitude worse than other storage alternatives. Even when storing 90% sparse data in relation *tomod10*, performance is still about a factor of five worse. Storing only one in ten cells already more than offsets the storage overhead in relations. The co-ordinates stored in the relation take 6 bytes for each 1 byte cell. Yet only relation

tomod2, with a density of 2%, and BLOB based storage are roughly comparable with storage in a specialised DBMS, but still RasDaMan is about a factor of two faster for all selectivities.

This also holds for small selectivities, where the relational system can use an index on one co-ordinate to reduce the amount of data fetched from disk. The bad performance of BLOB based storage, when compared with RasDaMan, came unexpected. It seems that access to a large number of 1-D intervals of data in a BLOB is an expensive operation. This has to be done, whenever the array stored is larger than the main memory of the client, as common in applications. Overall, for subselect operations on multidimensional arrays a specialised DBMS proved to be the most efficient solution in all tests.

5.2 Roll-Up Queries

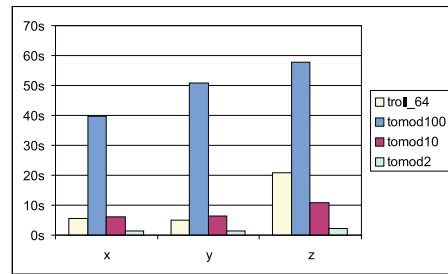


Figure 2. Roll-up.

Figure 2 compares the roll-up query in RasDaMan with arrays of different densities stored in relations. Roll-ups are done along all three axes. Again the 100% dense relation *tomod100* has significantly worse performance than the other measurements. For the roll-up operation, relational storage can compete with RasDaMan already at a density of 10%. One reason is, that the restriction to seven 2-D slices is a 1-D restriction with a very small selectivity, thereby enabling the RDBMS to use an index for efficiently retrieving the set of relevant tuples.

The spatial index of RasDaMan, on the other hand, always retrieves tiles with a thickness of 40 in this dimension. An adapted tiling strategy would give better results for this query. For relations with a sparsity of 98%, relational storage is significantly faster than RasDaMan on dense arrays.

5.3 Content Based Selection Queries

In Figure 3, performance results for the content based selection queries are given. Queries 1, 2, 3 select one, two, three images using a condition based on *some_cell*, and queries 4, 5, 6 select one, two, three images using

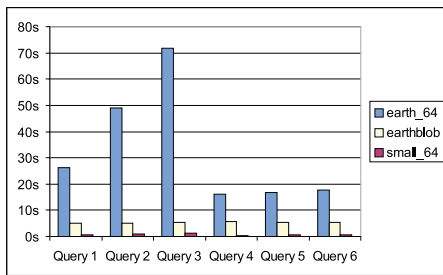


Figure 3. Content based selection.

`all_cell`. Results are also shown for storage in BLOBs (*earthblob*). The tests were executed for relational storage, but each query took more than two hours to execute on a relation with 100% density, which is two orders of magnitude worse than the other storage techniques.

RasDaMan, when executing the queries, first accesses all images in the earth collection tile by tile. For each tile the comparison operation is evaluated and on the result the condense operation `some_cell` or `all_cell` is applied. If the condense operation returns true the relevant area of this image is transferred to the client.

The selection of images is faster using BLOB based storage compared to *earth_64*. In our implementation, whole BLOBs are transferred to the client and the condition is evaluated there using a simple 1-D iteration directly implemented in C++ without consideration for the multidimensional nature of the arrays. This more efficient, but much less flexible, implementation of condition evaluation is one reason for the performance advantage. In the test environment, all processes were running on the same machine. In a network configuration, RasDaMan only would have to transfer the result to the client, but the BLOB implementation would have to transfer the whole data. This would make the BLOB solution much slower.

If the condition is evaluated only on a part of the image, the same effect can be observed. RasDaMan is much faster, because less data is read from secondary storage to evaluate the condition. In the query `small_64`, the condition is evaluated only on 1.6% of the array. This makes RasDaMan much faster than BLOB storage, because only this part has to be read from disk.

6. Conclusions

The aim of this work was to compare different storage techniques for multidimensional array data using queries from typical application areas. For scientific applications, subselects and content based selection queries were evaluated. Considering that dense arrays are common in this application area, modelling array data as relations is several orders of magnitude slower than alternative solutions.

Modelling arrays as relations can only compete, if data with a density of less than 2% is stored, which is not typical in this application area.

Storage in BLOBs delivers worse performance for subselect queries on arrays than a specialised DBMS. For some queries, BLOBs can deliver comparable performance, if the database client is connected to a local server. In a networked environment, a BLOB based solution has to transfer the whole data to the client whereas an array DBMS processes data on the server and only ships the result of a query. Queries on BLOBs have to be explicitly programmed in C++ resulting in 50 to 100 lines of code even for simple queries. If scientific applications are implemented on top of a DBMS and explorative analysis is desired, a specialised DBMS for multidimensional arrays is the most efficient solution both regarding performance and implementation effort.

To summarise, a specialised DBMS for multidimensional arrays such as RasDaMan offers better performance storing arrays than alternative storage techniques. Furthermore, a declarative query language based on SQL and augmented for array handling enables users to do interactive explorative data analysis. Implementing applications is possible with less effort if array functionality is supported in the DBMS. To compete with relational systems in OLAP applications dealing with sparse data, scientific array functionality would have to be extended with specialised constructs for OLAP operations.

Acknowledgements

The authors would like to thank Andreas Dehmel, Paula Furtado and Roland Ritsch, all working in the RasDaMan team at FORWISS.

References

- [1] M. Blaschka, C. Sapia, G. Höfling, and B. Dinter. Finding your way through multidimensional data models. In *Proc. International Workshop on Data Warehouse Design and OLAP Technology (DWDOT, in connection with DEXA'98)*, Vienna, Austria, Aug. 1998.
- [2] P. Furtado and P. Baumann. Storage of multidimensional arrays based on arbitrary tiling. In *Proceedings of the 15th International Conference on Data Engineering*, Sydney, Australia, Mar. 1999.
- [3] N. Widmann and P. Baumann. Towards comprehensive database support for geoscientific raster data. In *Proceedings of the ACM-GIS 97, Las Vegas, Nevada, USA*, Nov. 1997.
- [4] N. Widmann and P. Baumann. Efficient execution of operations in a dbms for multidimensional arrays. In *Proceedings of the SSDBM 98, Capri, Italy*, July 1998.