

Data Cube Compression with QuantiCubes

Pedro Furtado and Henrique Madeira
CISUC/DEI

Abstract: Data warehouses typically store a multidimensional fact representation of the data that can be used in any type of analysis. Many applications materialize data cubes as multidimensional arrays for fast, direct and random access to values. Those data cubes are used for exploration, with operations such as roll-up, drill-down, slice and dice. The data cubes can become very large, increasing the amount of I/O significantly due to the need to retrieve a large number of cube chunks. The large cost associated with I/O leads to degraded performance. The data cube can be compressed, but traditional compression techniques do not render it queriable, as they compress and decompress reasonably large blocks and have large costs associated with the decompression and indirect access. For this reason they are mostly used for archiving. This paper uses the QuantiCubes compression strategy that replaces the data cube by a smaller representation while maintaining full queriability and random access to values. The results show that the technique produces large improvement in performance.

1. Introduction

The amount of information stored and analyzed in data warehouses can be very large. Data cubes are frequently used to materialize views of data sets for Online Analytical Processing (OLAP). They can be very large and must be stored and retrieved from disk in costly I/O operations.

The multidimensional data cube representation uses fixed-size cells and has the advantages of full queriability and random access, as the position of any value is directly computable from the offset on each dimension axis. The major drawback of this representation is that it can occupy a lot of space, causing large access overheads. This has lead us to investigate compressed representations of normal data cubes that would reduce the number of I/O operations needed without obstructing queriability and random access.

Compressing the data cube has important advantages. The most obvious one is that it saves space (between 50% and 80% when QuantiCubes is used). However, the largest advantages are consequences of the smaller space usage. Whenever the memory is scarce compared to the size of the data sets composing the workload, those savings can mean less disk swapping, as more data fits in memory buffers and cubic caches. An even more important effect is that the most time consuming queries over data cube chunks stored on disk become almost 2 to 5 times faster simply because only one half to one fifth of the data has to be loaded from disk (chunk storage organization of data cubes on disk is discussed in [6]).

Traditional compression techniques are not useful in this context. They achieve large compression rates that are very useful for archiving (e.g. Arithmetic Coding, Lempel-Ziv, Huffman Coding, ...) [2,5,7,8]. However, the compressed data sets are not directly queriable without prior decompression of whole blocks, due to the variable-length of the compressed fields. The term “variable-length field” refers

to the usage of a variable number of bits to represent compressed values. Compression/decompression is done by blocks and require expensive access methods to locate the correct block. This indirect access feature accounts for a large overhead. The decompression overhead of most typical techniques is also significant, as it includes bit processing operations and access to auxiliary structures such as dictionaries or code trees to decode the bitstream.

In this paper we present the QuantiCubes technique, which packs reduced fixed-width representations of the individual cells to compress the data cube as much as possible for faster analysis, while maintaining the full queriability and random access properties of the uncompressed data cube. These results are only possible because QuantiCubes guarantees insignificant decompression overhead and maintains the same degree of random access and queriability of the uncompressed data cube, because it stores fixed-width compressed values. The technique is presented and tested in this paper.

The paper is organized as follows: section 2 presents the QuantiCubes technique. Section 3 presents experimental results and section 4 concludes the paper.

2. The QuantiCubes Technique

QuantiCubes transforms the normal data cube representation. The data is analyzed first to determine a fixed-width compressed coding that is optimal with respect to a set of error and space criteria. This is achieved using an adaptable quantization procedure. The data cube values are then coded and packed to achieve a very large compression rate. For example, a data cube cell occupying 4 bytes is compressed to the size of the bitcode that was determined for that cell (e.g. 9 bits). Finally, extremely fast decompression of individual values “on-the-fly” is achieved by unpacking and dequantizing the values with very fast, low-level operations.

2.1. Data Analysis

Quantization [4] is used in QuantiCubes to determine the set of representative values for an attribute. The crucial issue of accuracy is guaranteed by our new proposal for an “adaptable” quantization procedure. This adaptable procedure uses an accuracy constraint based in user-defined or preset error bounds during the determination of quantization levels. This way, the number of levels that are necessary is determined adaptively and attributes that cannot be estimated with large accuracy using a reasonable number of quantization levels can be left uncompressed. The new adaptable quantization algorithm is not discussed here for lack of space. A complete discussion of the algorithm is available in [1].

The specific OLAP context is important in the adaptable quantization procedure, because many fact attributes measuring quantities such as daily or periodic sales offer favorable cases for the replacement by a number of representative values, as they are often distributed around clusters and repeated values. The best results (most accurate and with less bits) are achieved when the data set is made of a small

set of distinct values that are repeated all the time. In that case the quantization levels coincide with distinct values and the bitcodes are simply a more compact representation of the values. In the extreme case, an attribute could have a small number of distinct values in a very large number of rows (e.g. 10M rows). Quantization also returns very good approximations when a significant fraction of the values are distributed around sets of clusters, in which case reconstruction values converge to the centers of distinct clusters. As more levels are added, the quantizer finds the most appropriate positions for the quantization levels to represent most values accurately.

Both the analysis and experimental results in [1] show that the approach guarantees accuracy and typically provides almost exact value reconstruction and precise query answers. This is an important issue, as the technique maintains the queriability of uncompressed data but is also able to guarantee high levels of accuracy for every query pattern and even in the reconstruction of individual values.

2.2. Reconstruction Array

As with many other compression techniques, an auxiliary structure must be accessed to convert between values and bitcodes. However, a small reconstruction array is used and a single direct access to a cell is required for decompression. Figure 1 shows an example that illustrates a reconstruction list. It is a simple array ordered by the reconstruction values with typical sizes between 64 and 8192 values, corresponding to bitcodes with 6 to 13 bits. In this list, bitcodes coincide with the index of the cell. The array is accessed to retrieve the bitcode (index in the array) or the reconstructed value (the value in each position of the array).

3.234	9.21	14.764	21.37	...	102.54
-------	------	--------	-------	-----	--------

Figure 1 – Illustration of a Reconstruction List

During compression, the reconstruction array is accessed to translate values into the bitcodes of the reconstruction values that are nearest to those values (the bitcodes coincide with the indices, as mentioned above). During decompression, the bitcodes are translated back to the corresponding reconstruction values.

2.3. Compression

Compression requires an access to the reconstruction array to retrieve the bitcode that replaces the value and a concatenation of that value into a word. Each word contains a small number of packed bitcodes and is sized to minimize the overhead of decompression [1].

The lookup in the reconstruction array requires a binary search of the array to determine the quantization level. The search must find the reconstruction value that is closest to the value that must be compressed and return the corresponding index. This

bitcode is then packed (concatenated) into the word using two low-level operations (shift = <<, OR=|, AND=&): $\text{bitcode}_i = ((\text{data_register} \ll \text{shift}_i) | \text{maskregister}_i);$

2.4. Decompression

The most crucial operation is decompression, as it is used for “on-the-fly” reconstruction of values during the processing of queries. It requires unpacking the bitcodes and a direct access to the reconstruction array, which is conveniently small to fit in the microprocessor internal cache after a few initial misses. Unpacking uses the operation: $\text{bitcode}_i = ((\text{data_register} \gg \text{shift}_i) \& \text{maskregister}_i);$
 The reconstruction value is retrieved using a direct access to the reconstruction list:

$\text{Reconstructed_value} = \text{Reconstruction_List}[\text{bitcode}_i];$

With this strategy, the unpacking operations are extremely fast.

3. Analysis and Evaluation of the Technique

We have measured 2 to 5 times improvement in access to the compressed data cube on disk, due to the much smaller number of accesses that is necessary to load the compressed cube, while also saving a large amount of space. This is shown in Figure 2, which compares the time taken to load a normal and a compressed data cube, when the compressed data cube has 1/4 of the size of the original data cube, as shown in the first column.

Size(MB) Normal/Cmprss	Cmprss	UnCmprss
100/25	11.837	49.431
200/50	23.434	104.01
400/100	48.54	212.158
600/150	73.814	333.631
800/200	112.99	495.07

Figure 2. Loading Time (secs)

In fact, the data cube does not need to be decompressed before operation or analysis even when in memory, as online decompression of individual values does not degrade the performance. This property saves a significant amount of memory buffer space. Assuming that Y values are operated upon in the OLAP query (we use the example Y =100 million values with 4 bytes each – 400MB - and 4 times compression rate), the following operations are necessary:

Uncompressed data cube:

4Y bytes loaded from memory (100 M values = **400 MB**)

+ Y × aggregation-related operations (100 M operations)

Compressed data cube:

Y bytes loaded from memory (100 M values = **100 MB**)

- + Y unpack operations (100 M ops) + M lookup operations (100 M ops)
- + Y × aggregation-related operations (100 M ops)

This discussion shows that operation on the compressed cube accesses a much smaller number of memory words, but the bitcodes must be unpacked and looked up. Predicting which of these operations is fastest is not trivial. It depends on the scheduling of instructions, the time taken to execute them and the use of internal caches by the microprocessor in the lookup functionality. However, by compiling and executing the code in two different systems, it was possible for us to conclude that the overhead incurred by the decompression is not larger than the overhead required to fetch the excess of uncompressed values. Figure 3 shows the time taken to operate on the compressed and uncompressed data cubes and also on an optimized version of the uncompressed data cube that uses loop unrolling [3] to try to match the speed of operation on the compressed data cube. This code was executed in a Pentium II-300Mhz processor.

Size(MB) Normal/Cmprss	Cmprss	UnCmprss	UnCmprss, Unrolled
25/6.25	0.601	1.352	0.610
50/12.5	1.132	2.744	1.192
100/25	2.353	5.498	2.354
150/37.5	3.455	8.232	3.555
200/50	4.607	11.026	4.736

Figure 3 – CPU Execution Time in Sum Operation

The adaptable quantization procedure produces an approximate data set. However, our new proposal for adaptable quantization allows the algorithm to find the appropriate number of levels that are necessary. Figure 4 shows the relative error (average, maximum and standard deviation) of the reconstruction of individual values taken by the attribute “Account Balance” of a bank data set as more bits are added to the quantizer (each additional bit doubles the number of levels available). This attribute had irregular distribution. The result shows that the strategy is able to adapt the number of levels until the error constraints are met. In this case, the quantity was well estimated using 10 to 12 bits.

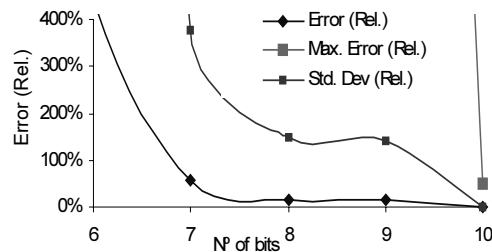


Figure 4-Adaptable Quantization of Attribute Account_Balance

Our experimental evaluation included other experiments that are not shown for lack of space. A more complete discussion and set of experiments is included in [1]. Those results include the comparison of decompression overhead with other compression techniques, showing that QuantiCubes is at least 10 times faster. The compression overhead was also tested experimentally. Although compression is slower than decompression, we have determined experimentally that the time taken to compress and store the data with QuantiCubes is not larger than the time taken to store the uncompressed data, because the compressed data cube is much smaller.

Conclusions and Future Work

In this paper we have proposed a technique (QuantiCubes) which is able to compress data cubes 2 to 5 times and still maintain complete queriability and random access. We have shown that the decompression overhead is completely insignificant, so that “on-the-fly” decompression of any number of values is possible in any situation without significant degradation of performance. The technique is also able to deliver 2 to 5 times faster operation than the uncompressed data. The fast operation of the technique is analyzed in detail and confirmed using experimental results. The technique is used in the context of analysis and operation on the data cube. It is not intended to replace data compression techniques in their archiving or longer-term disk storage function, as they achieve very large compression rates using a variable-width, exact representation of the data set.

References

1. Furtado, Pedro “Accurate Reduced Representations of Multidimensional Fact-Like Data Sets”, PhD thesis, Universidade de Coimbra, 2000.
2. J-L Gaily and M. Adler. Zlib home page. <http://quest.jpl.nasa.gov/zlib/>.
3. J. Hennessy and D. Patterson, “Computer Architecture: A Quantitative Approach”. Morgan Kaufmann, 1990: ISBN 1-55860-069-8.
4. S.P. Lloyd. Least Squares Quantization in PCM. IEEE Transactions on Information Theory, IT-28:127-135, March, 1982.
5. M. Nelson, J-L Gaily, “The Data Compression Book”, 2nd edition, 1996 - M&T Books, ISBN 1-55851-434-1.
6. S. Sarawagi and M. Stonebraker, “Efficient Organization of Large Multidimensional Arrays”, in Proceedings of the 11th Annual IEEE Conference on Data Engineering (ICDE’94), Houston, Texas, 1994.
7. Welsh, Terry, “A Technique for High-Performance Data Compression” IEEE Computer, Volume 17, N° 6, June 1984, pages 8-19.
8. J. Ziv, Lempel, “A Universal algorithm for sequential data compression”, IEEE Transactions on Information Theory, Volume 23, N° 3, May 1977, pages 337-343.