Poster Paper

Model Kit for Lightweight Data Compression Algorithms

Juliana Hildebrandt, Dirk Habich, Patrick Damme, Wolfgang Lehner
Technische Universität Dresden
Database Systems Group
01189 Dresden, Germany
firstname.lastname@tu-dresden.de

ABSTRACT

Modern database systems are very often in the position to store and efficiently process their entire data in main memory. Aside from increased main memory capacities, a further driver for in-memory database systems has been the shift to a column-oriented storage format in combination with lightweight data compression techniques. In recent years, a lot of lightweight data compression algorithms have been developed to efficiently support different data characteristics. Therefore, database systems should include a large number of these algorithms. To enable this, we introduce our novel modularization concept including our model kit implementation for lightweight data compression algorithms.

1. MOTIVATION

With an ever increasing amount of data in almost all application domains, the storage requirements for database systems grow quickly. In the same way, the pressure to achieve the required processing performance increases, too. To tackle both aspects in a uniform way, data compression plays an important role. On the one hand, data compression drastically reduces storage requirements. On the other hand, data compression also is the cornerstone of an efficient processing capability by enabling "in-memory" technologies. As shown in different papers, the performance gain of in-memory data processing is massive because the operations benefit from its higher bandwidth and lower latency [1].

Especially for the use in in-memory database systems, a variety of lightweight compression algorithms have been developed. These algorithms achieve good compression rates with little computational effort for compression as well as decompression. The main classes of lightweight data compression techniques are dictionary compression (DICT), delta coding (DELTA), frame-of-reference (FOR), null suppression (NS), and run-length encoding (RLE) [2, 3]. The algorithms in each class evolve further and the development activities increase over the years, whereas the concept of the classes are interweaved in new algorithms.

Generally, this multitude of algorithms exists because it is impossible to design an algorithm that automatically produces optimal results for any data. In order to support and to implement a wide range of these algorithms in a database system, a unified approach for the specification or engineering is desirable. In our current research, we have focused on that aspect by developing a novel compression scheme consisting of a few number of modules. Our poster presentation at EDBT comprises an in-depth explanation of this compression scheme. As we are going to show, our compression scheme is quite suitable to modularize a variety of lightweight data compression algorithms in a systematic manner. That means, our approach offers an efficient and an easy-to-use way to describe, to compare, and to adapt lightweight data compression techniques. Furthermore, we want to introduce our model kit implementation that is founded on that compression scheme.

2. MODULARIZATION CONCEPT

Our novel compression scheme consists of four main modules as shown in Figure 1; the arrows indicate the data flows. The input is a sequence of uncompressed values, the output is a sequence of compressed values. Our whole scheme is a `Recursion` module. The first module in each `Recursion` is a `Tokenizer` splitting the input sequence in finite subsequences or single values. For that, a `Tokenizer` can be parametrized with a calculation rule. Its output is a token, a finite sequence of values that serves as input for our second module, the `Parameter Calculator`. Parameters are often required for the encoding and decoding. This module follows special rules (parameter definitions) for the calculation of several parameters. We summarize different data structures like single values calculated from sequences, dictionaries or vectors as parameters belonging to a token. Our third module is the `Encoder`, which can be parametrized with a

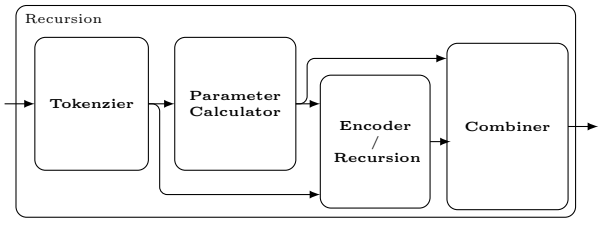

Figure 1: Modularized compression scheme

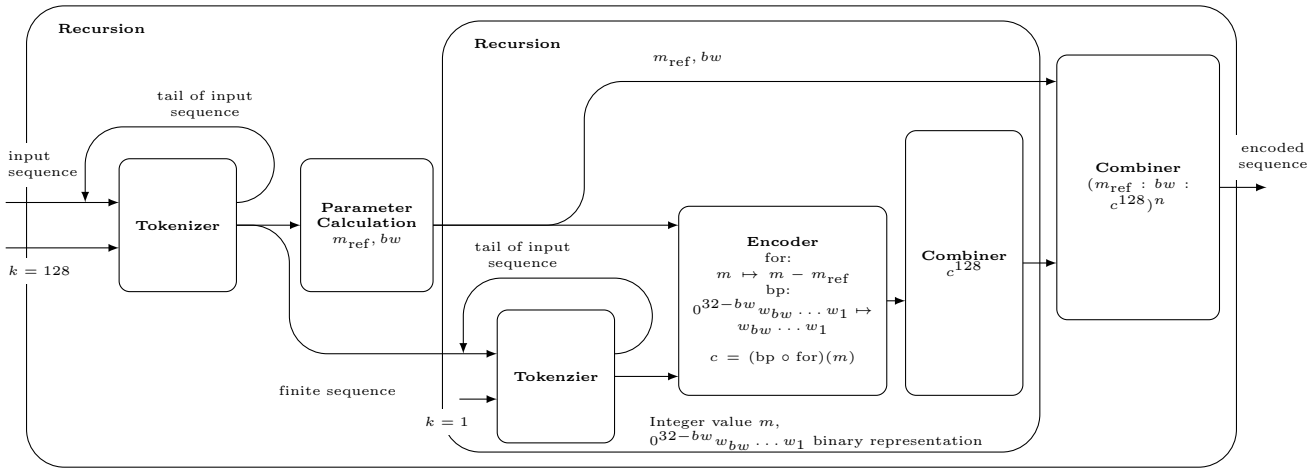

Figure 2: Modularized scheme for the frame-of-reference technique with binary packing

calculation rule for the processing of an atomic input value, whereas the output of the `Parameter Calculator` is an additional set of parameters needed for the calculation inside the `Encoder`. The fourth module is the `Combiner`. It determines how to arrange the output of the `Encoder`. For a more sophisticated hierarchical data partitioning and parameter calculation, we are able to replace the `Encoder` with a `Recursion` module.

3. POSTER PRESENTATION DETAILS

Our EDBT poster presentation comprises an in-depth explanation of the compression scheme. In particular, we want to show the applicability by the transcription of a variety of lightweight data compression algorithms. Due to space constraints, we only present one example transcription in this paper: *semi-adaptive frame-of-reference algorithm with binary packing for the compression of positive integer values*. For each set of 128 values, the algorithm calculates the minimum value as reference value. Then, each single integer value is mapped to the offset to the reference value at the logical level. This technique leads to smaller numbers. Then, all 12 offset values are encoded with the same bit width on the physical level. That means, the algorithm has to calculate it. For each compressed sequence of 128 values, we have to store the reference value and the bit width as meta data. Otherwise we are not able to decode the values.

Figure 2 shows the algorithm in our novel compression scheme, which can be directly mapped to our model kit implementation. The first `Tokenizer` is parametrized with a calculation rule that determines that the `Tokenizer` has to output the first 128 32-bit integer values of the input sequence. The tail of the sequence serves as further input for the `Tokenizer` and is processed in the same way in a next step. The `Parameter Calculator` determines the minimum of the 128 values as reference value m_{ref} and the needed bit width bw to encode all 128 offset values. Instead of an `Encoder` we use a `Recursion` module. Inside that recursion, we have a `Tokenizer` outputting single integer values. Logically, we have a `Parameter Calculator`. But at that level, we do not need to calculate any parameter here. The `Encoder` manages the logical level of encoding as well as the bit level. It uses the reference value m_{ref} and the calculated bit width bw as parameters. At the logical level it calculates with the function *for* the offset to the common reference value for each of the 128 values. At the bit level, it determines the binary representation of the offset with the help of the bit width. The inner `Combiner` concatenates all physical representations of the 128 offset values to a compressed sequence. Out of the inner `Recursion` the outer `Combiner` concatenates the compressed sequence with the physical representation of m_{ref} and bw as long as all input has been processed.

4. MODEL KIT AND CONCLUSION

Based on our novel modularized scheme, we also developed an appropriate model kit on the implementation level using C++. Our defined modules are available as building blocks, which can be parameterized with certain calculation rules. The building blocks can be orchestrated to data flows, so that complete lightweight data compression algorithms can be realized. Next, we want to optimize the building blocks, so that the algorithms can be efficiently executed. Furthermore, we want to use the model kit to integrate a large number of algorithms in a database system.

To summarize, in our EDBT poster presentation, we introduce our novel developed compression scheme for lightweight data compression algorithms. In particular, we want to show that our approach offers an efficient and an easy-to-use way to describe, to compare, to adapt, and to implement lightweight data compression techniques.

5. REFERENCES

[1] P. A. Boncz, S. Manegold, and M. L. Kersten, "Database architecture optimized for the new bottleneck: Memory access," in *VLDB*, 1999, pp. 54–65.

[2] H. K. Reghbati, "An overview of data compression techniques." *IEEE Computer*, vol. 14, no. 4, pp. 71–75, 1981.

[3] D. J. Abadi, S. R. Madden, and M. C. Ferreira, "Integrating compression and execution in column-oriented database systems," in *In SIGMOD*, 2006, pp. 671–682.