Revisiting Data Compression in Column-Stores

Alexander Slesarev, Evgeniy Klyuchikov, Kirill Smirnov, George Chernishev
{alexander.g.slesarev,evgeniy.klyuchikov, kirill.k.smirnov, chernishev}@gmail.com

Saint-Petersburg University, Saint-Petersburg, Russia

MEDI 2021
June 23, 2021

Background I: compression in DBMSes

Types [ABH13, HAB09]:

- Light-weight
- Heavy-weight

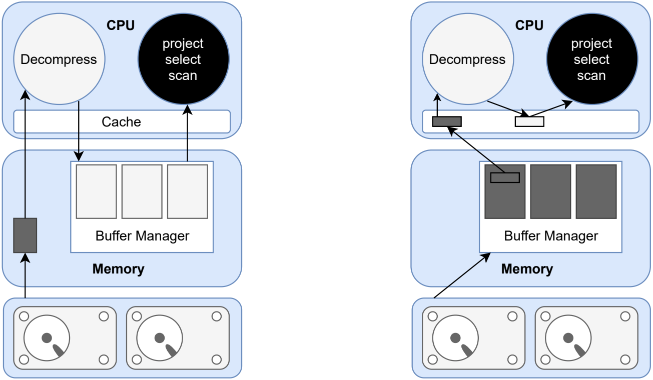
Goals:

- Speed up query by reducing disk read time
- Reduce data volume on disk

Benefits in column-store:

- One attribute — one data type
- Encode multiple elements at a time using SIMD

Background II: compression schemes



(a) I/O-RAM

(b) RAM-CPU

Figure 1: Compression implementing approaches, adapted from [ZHNB06]

Background III: compression in column-stores

Pioneers of modern column-stores: light-weight (RLE, FoR, differential, etc) algorithms superior to heavy-weight (BZIP, ZLIB).

Reasons:

- $cost_{light}(decoding\_effort) + cost_{light}(disk\_read) < cost_{heavy}(decoding\_effort) + cost_{heavy}(disk\_read)$
- other benefits: operating on compressed data directly, cache-friendliness, etc

But what now? Fifteen years have passed — time to reevaluate:

- CPU, RAM, and disk performance have considerably advanced;
- novel compression algorithms have appeared;
- SIMD-enabled versions of existing algorithms have appeared.

Research questions

- RQ1: Are heavy-weight compression schemes still inappropriate for disk-based column-stores?
- RQ2: Are new light-weight compression algorithms better than the old ones?
- RQ3: Is there a need for SIMD-employing decompression algorithms in case of a disk-based system?

Considered compression methods

- Light-weight:
  - Regular: PFOR [ZHNB06], VByte
  - SIMD-enabled: SIMD-FastPFOR [LB15], SIMD-BinaryPacking [LB15]
- Heavy-weight:
  - Brotli [AFF$^+$18]

PosDB: overview [CGG^{+}18]

A distributed disk-based column-store for research purposes:

- Relies on Volcano block-based iterator model.
- Columnar: operators exchange not only data, but also positions (PosDB).
- Disk-based: data $>>$ main memory.
- Distributed: has send & receive operators. Not mediator-based, but "true" distribution of data and queries.
- Parallel: any operator sub-tree can be executed in a separate thread.

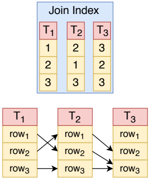


Figure 2: Example of join index

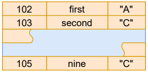


Figure 3: Example of tuple representation

PosDB: query plans

```
SELECT sum(lo_revenue), d_year, p_brand1
FROM lineorder, date, part, supplier
WHERE    lo_orderdate = d_datekey
     and lo_partkey = p_partkey
     and lo_suppkey = s_suppkey
     and p_category = 'MFGR#12'
     and s_region = 'AMERICA'
GROUP BY d_year, p_brand1
ORDER BY d_year, p_brand1;
```

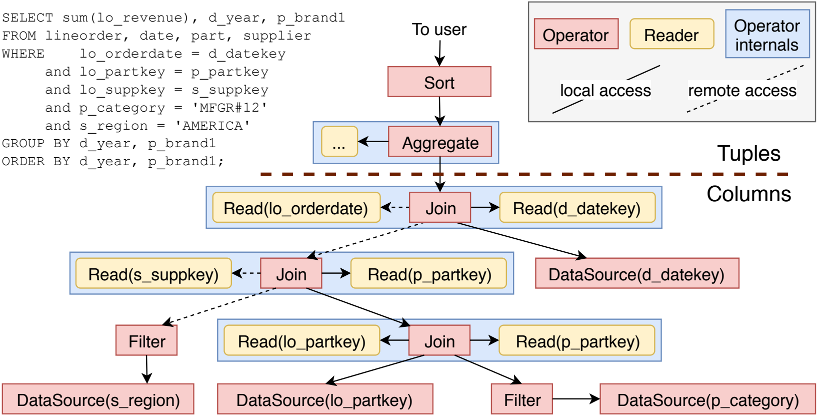
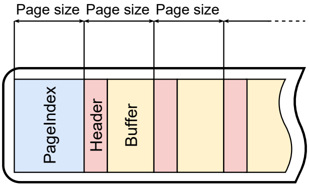
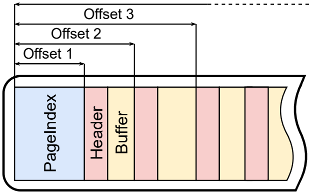


Figure 4: Query plan example

PosDB: file formats



(a) Uncompressed file



(b) Compressed file

PosDB: access to compressed data

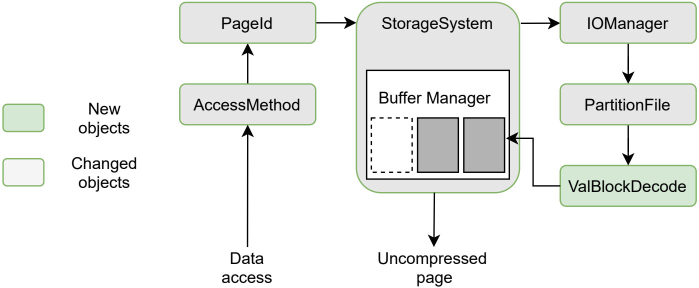


Figure 6: Compressed page receive in case it was not cached

Configuration

Experiments were run with:

- PosDB v0043bba9, single-node configuration, buffer manager 16000 pages (1 GB)
- Hardware: Inspiron 15 7000 Gaming(0798), 8GiB RAM, Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz, TOSHIBA1TB MQ02ABD1
- Software: Ubuntu 20.04.1LTS, 5.4.0-72-generic, g++ 9.3.0

We used Star Schema Benchmark with Scale Factor 50 (16 GBs).

Evaluation I: compression rates

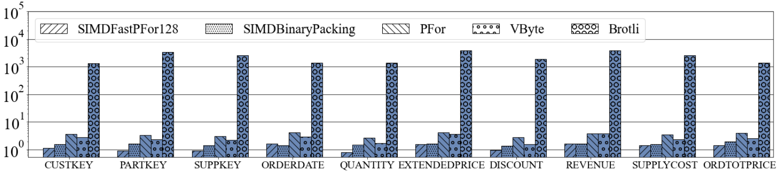


Figure 7: Compression time (Seconds)

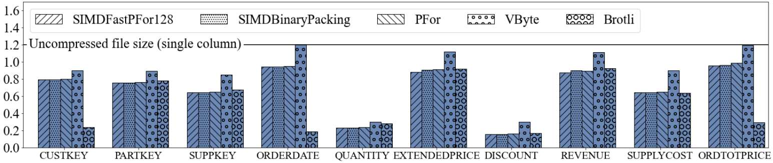


Figure 8: Compressed column sizes (Gigabytes)

Evaluation II: run time performance

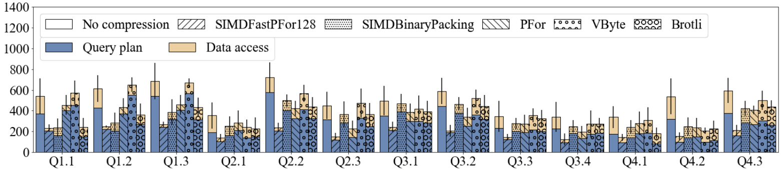


Figure 9: System run time break down for "parallel" scenario (Sec.)

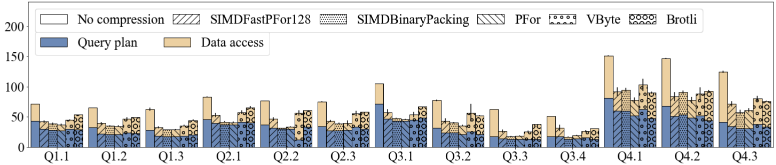


Figure 10: System run time break down for "sequential" scenario, (Sec.)

Evaluation III: performance in depth

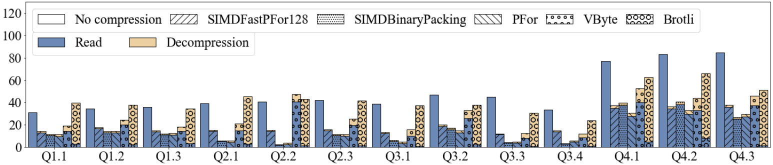


Figure 11: IO thread action breakdown (Seconds)

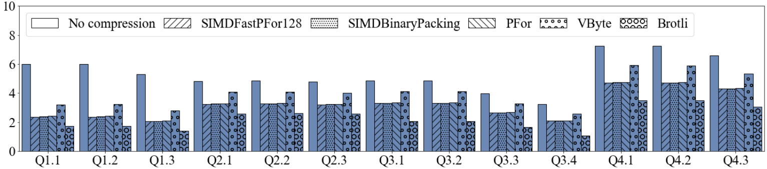


Figure 12: Total volume of data read by query (Gigabytes)

Conclusion

- RQ1: Are heavy-weight compression schemes still inappropriate for disk-based column-stores?
  - Largely yes. Loses to light-weight approaches, but still can give 20% speed improvement over the uncompressed case. Makes sense to use them to save disk space, if data is static and long-living.
- RQ2: Are new light-weight compression algorithms better than the old ones?
  - We can't definitely conclude that there is progress (beneficial to DBMSes) in light-weight compression schemes, aside from the appearance of SIMD-enabled versions.

    Also, VByte is BAD.
- RQ3: Is there a need for SIMD-employing decompression algorithms in case of a disk-based system?
  - Yes, since decompression happens in a dedicated thread. Relative decompression costs are still high.

References

Daniel Abadi, Peter Boncz, and Stavros Harizopoulos.
The Design and Implementation of Modern Column-Oriented Database Systems.
Now Publishers Inc., Hanover, MA, USA, 2013.

Jyrki Alakuijala, Andrea Farruggia, Paolo Ferragina, Eugene Kliuchnikov, Robert Obryk, Zoltan Szabadka, and Lode Vandevenne.
Brotli: A general-purpose data compressor.
ACM Trans. Inf. Syst., 37(1), December 2018.

G. A. Chernishev, V. A. Galaktionov, V. D. Grigorev, E. S. Klyuchikov, and K. K. Smirnov.
PosDB: An architecture overview.
Programming and Computer Software, 44(1):62–74, Jan 2018.

Stavros Harizopoulos, Daniel Abadi, and Peter Boncz.
Column-oriented database systems, VLDB 2009 tutorial., 2009.

D. Lemire and L. Boytsov.
Decoding billions of integers per second through vectorization.
Softw. Pract. Exper., 45(1):1–29, January 2015.

M. Zukowski, S. Heman, N. Nes, and P. Boncz.
Super-scalar RAM-CPU cache compression.
In *ICDE'06*, pages 59–59, April 2006.