Data Compression in PostgreSQL

postgrespro.com

Michael Zhilin
m.zhilin@postgrespro.com

About me

Michael Zhilin (@mizhka)

- **Performance Engineer since 2008**
- **Performance Team Lead at PostgresPro since 2020**

e-mail:
m.zhilin@postgrespro.com
mizhka@gmail.com

Agenda

- Compression in general
- Why may database compression be useful?
- Built-in PostgreSQL compression
- Advanced data compression options for PostgreSQL
- Use cases and comparison of each technique’s key advantages
- Q&A session

Compression: terms

- Information entropy [1]
  - Measure of data randomness
- Lossless compression
- Key parameters
  - Compression ratio
  - Compression and decompression speed

[1] https://en.wikipedia.org/wiki/Entropy_(information_theory)

Compression: history

- Shannon-Fano (1948-1949) [1]
  - Probability of symbols
- Huffman encoding (1954) [2]
  - minimum encoding
  - binary tree based on frequencies
  - a.k.a. prefix code

[1] https://en.wikipedia.org/wiki/Shannon%E2%80%93Fano_coding

[2] https://en.wikipedia.org/wiki/Huffman_coding

Compression: history

- Lempel-Ziv-Welch (1978-1984) a.k.a. LZ1/LZ2 [1]
  - gif
- Deflate (1991) [2]
  - png, zip, gzip and others
- LZO (1996)
  - very fast decompression: read-only file systems
- LZ4 (2011), Zstandard (2016)

[1] https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch

[2] https://en.wikipedia.org/wiki/Deflate

Compression: algorithms

lzbench [1] & Silesia corpus

Algorithm	Description	Compression	Decompression	Ratio
deflate	old, but good	5-100MBps	10-200MBps	2.8
lzo	quick deco	8MBps	850MBps	2.8
lz4	fastest	780MBps	4500MBps	2.1
zstd	good balance	480MBps	1200MBps	2.8

[1] https://github.com/inikep/lzbench

Why may DB compression be useful?

- Saving disk space
    - This is essential for multi-TB databases
- Saving memory, improving caching
- Reducing amount of disk I/O operations
    - Improves throughput (more queries per second)
    - Improves latency (faster response times)

Why may DB compression be useful?

Trade-off

- **Performance degradation**
  - **Requires more CPU resources and time**
- **Compatibility issues**

Built-in Postgres compression: what to compress?

- Database instance files
  - relations (tables, indexes, TOAST)
  - write-ahead logs (recovery, replication, backups)
- Others
  - backup files (wal-g, probackup, pg_backrest)
  - logical dumps (pg_dump)

Built-in Postgres compression: tuples

- Tuple fields are compressed if their size > 2K bytes
  - In-line storage for short compressed fields
  - TOAST storage for big compressed fields
- Algorithm
  - PGLZ
  - LZ4 since PostgreSQL 14

Built-in Postgres compression: B-Tree index

- B-Tree index key deduplication
  - since PostgreSQL 12
  - since PostgresPro 10
- Storing posting lists of TIDs

Built-in compression: WAL FPI

- wal_compression=on
- Since PostgreSQL 9.5
- Only “full-page image” compression
- Algorithm
  - PGLZ
  - LZ4 since PostgreSQL 15 (the upcoming release)

Built-in compression: what’s missing…

- Heap value deduplication [1]
- Index key compression
- Fast TOAST

[1] https://www.postgresql.eu/events/pgconfeu2019/sessions/session/2671/slides/263/Data_Compression_in_PostgreSQL_and_its_future_noscript.pdf

Advanced data compression options

Columnar storage

- GreenPlum & ZedStore (fork) by GreenPlum
- Citus Columnar & cstore_fdw (extension) by Citus
- Various compression options:
  - Append-only optimizations
  - lz4, zstd, zlib, rle

There is set of limitations (check documentation)
No index compression

Advanced data compression options

Compressed filesystems

- OpenZFS (Zettabyte filesystems) [1]
- lz4, zstd, tuning parameters
- Transparent for database

Copy-on-write: possible slowness and bad scalability
Requires configuration skills and tuning for database engines

[1] https://openzfs.readthedocs.io/en/latest/performance-tuning.html#postgresql

Advanced data compression options

PostgresPro CFS [1]

- Designed for PostgreSQL page-organized files (tables, indexes)
- Transparent page compression
- Easy configuration, separate tablespace
- lz4, zstd, zlib, pglz

Brings simplicity and power of compression in one shot.
Available in Postgres Pro Enterprise 9.6+

[1] https://postgrespro.com/docs/enterprise/13/cfs-usage

Use cases and comparison

#1: Small-size deployment

- <500GB database size
- <500 tps / qps
- <16 vCPU
- <64 GB RAM

Built-in compression is a good choice!

Use cases and comparison

#2: Middle-size deployment

- <20TB database size
- <5000 tps / qps
- <128 vCPU
- <512 GB RAM

Consider advanced techniques to speed up queries and save storage space.

Use cases and comparison

#3: Huge deployment

- **>20TB of various data**
- **>5000 tps / qps**

It is strongly recommended to use compression techniques!

Use cases and comparison

#4: lots of files stored in the database (e.g. PDF files or photos)

- **Tuple compression and TOAST are used**
- **Compression rate is good, but performance is poor.**

Alternative: store files outside the database and keep only meta information in database tables.

Use cases and comparison

#5: full selections done on a small number of columns (a kind of analytics DB)

- No indexes on columns
- No built-in compression and deduplication

Columnar store is the best choice.

Use cases and comparison

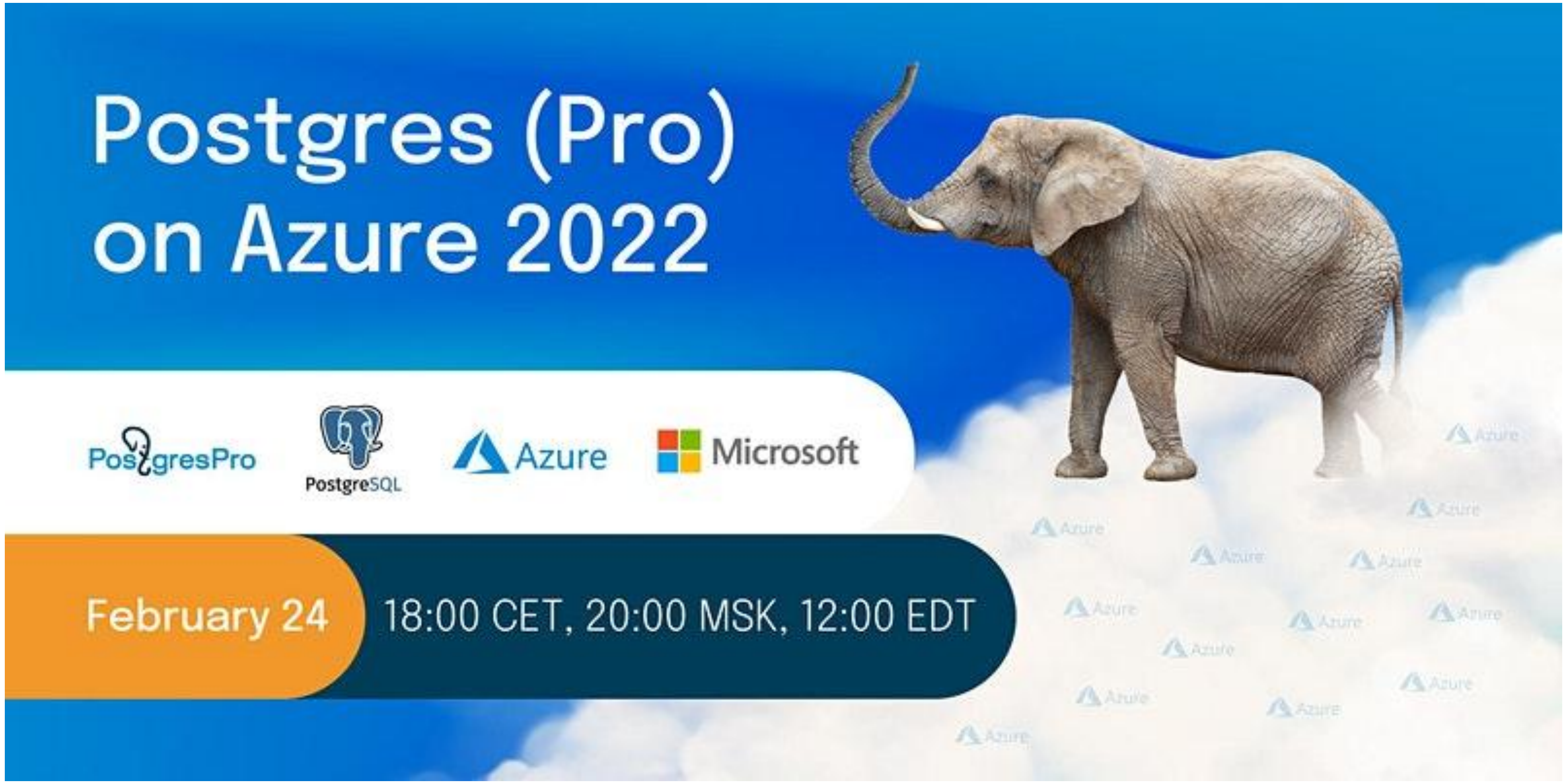
#6: encrypted data

- Encryption increases the entropy of data
- No built-in encryption mechanisms

Compression should be done first, before encryption.

Next event: February 24, 2022

https://www.eventbrite.co.uk/e/postgres-pro-on-azure-2022-tickets-247481954187



Postgres (Pro) on Azure 2022

February 24 18:00 CET, 20:00 MSK, 12:00 EDT

Questions?

postgrespro.com