

FOSS4G
Prizren, 2023

SOZip: using directly large (geospatial) compressed files in a ZIP archive!

Even Rouault
SPATIALYS

June 28th 2023



Why SOZip ?

- Number of popular Geospatial formats don't come with native compression:



FlatGeobuf



- And despite growing storage capabilities, we always need compression:
 - Lower costs
 - Faster download times

Why SOZip ?

Which options for formats that don't support compression:

- Add one inside the format !
 - Can be fine-tuned to the specificities of the format, e.g. SQLite3 has a CEROD (Compressed and Encrypted Read-Only Database) *proprietary* extension to compress individually SQLite pages
 - But breaks forward compatibility with existing readers
 - Additional implementation complexity
 - Must be done on a format case-by-case basis
- Do not modify the file format, but put it inside an archive format: .zip, .tar, .7z, etc.

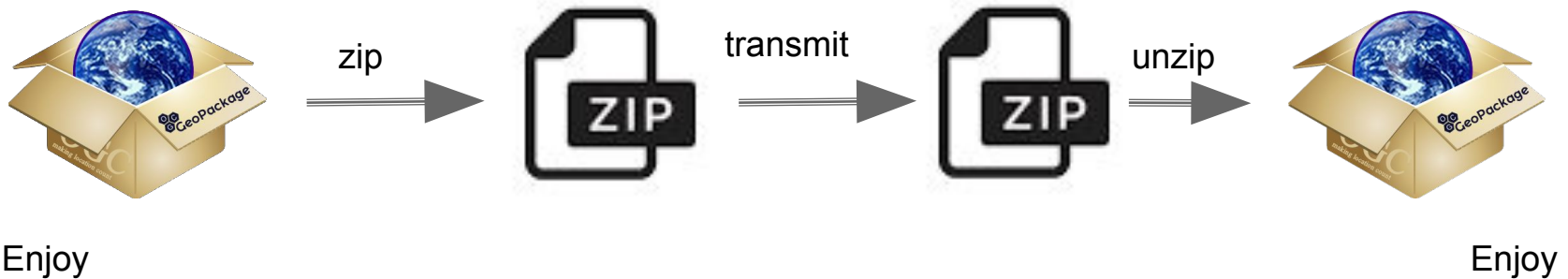
Let's put them in a ZIP!

- ZIP is ubiquitous.
- Easy to do ! ZIP compression and decompression is a one-click operation in modern operating systems
- Compresses well
- Example with a dataset with the 3.2 millions footprint polygons of buildings of New Zealand with 13 attributes each

Format	Uncompressed size	.zip size	Compression ratio
GeoPackage	1.666 GB	480 MB	3.47
FlatGeoBuf	1.826 GB	455 MB	4.01
Shapefile	2.948 GB	396 MB	7.44

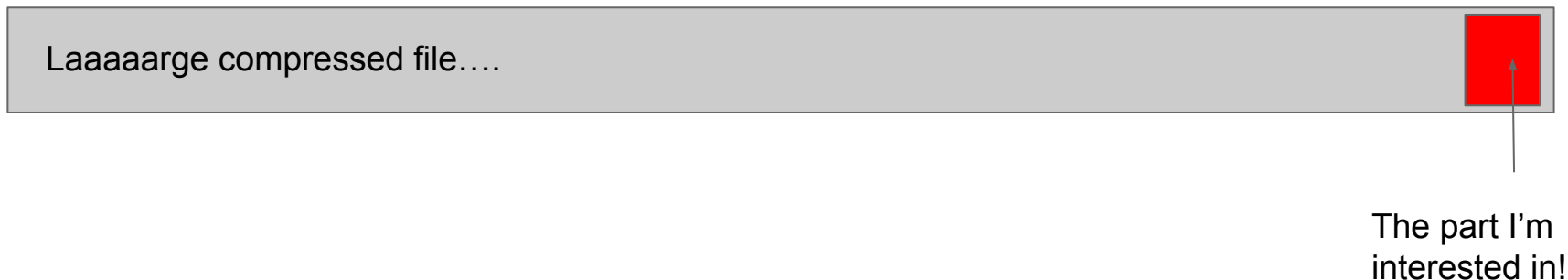
Let's put them in a ZIP!

- ... but ZIP is thought as mostly a transport/archiving operation
- A file once Zipped is temporarily unusable



We can do better

- ZIP has an indexing mechanism to locate the start of each compressed file within an archive
(contrary to the .tar format)
- So it is possible for a smart enough reader to read a file !
- Yes...but if you read it from its beginning to the end (or up to the point you're interested in in the file)



- The /vsizip/ virtual file system in GDAL can already do that.

A few insights on Deflate

- ZIP historical and widely used codec is Deflate
- Standardized as <https://datatracker.ietf.org/doc/html/rfc1951>
- Overview at <https://en.wikipedia.org/wiki/Deflate>
- Deflate = LZ77 dictionary-based + Huffman compression
- LZ77: sliding window of 32 kB over uncompressed data, with emissions of:
 - Literal bytes [0,255] when no repetition found
 - (length, distance) tuples when repetitions are found

Uncompressed stream:

This is a long sentence isn't it?

Compressed stream:

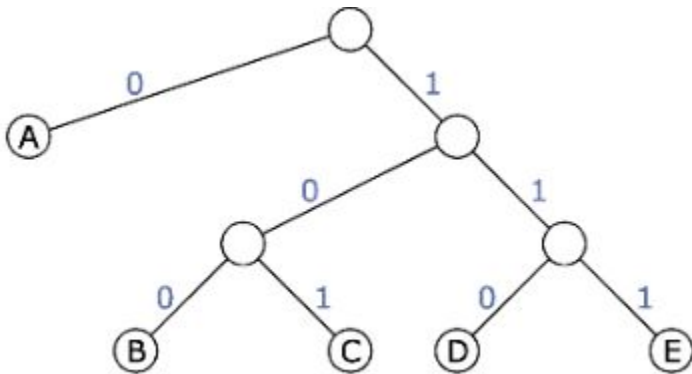
This [3,3]a long sentence[3,19]n't it?



The diagram illustrates the LZ77 sliding window mechanism. It shows two lines of text. The top line is the uncompressed stream: "This is a long sentence isn't it?". The bottom line is the compressed stream: "This [3,3]a long sentence[3,19]n't it?". The words "is" and "isn't" in the uncompressed stream are highlighted in red and blue, respectively. In the compressed stream, the tuples [3,3] and [3,19] are highlighted in green. Two arrows point from the green boxes in the compressed stream to the corresponding words in the uncompressed stream: one from [3,3] to "is" and one from [3,19] to "isn't".

A few insights on Deflate

- Huffman code trees used to encode literals bytes/lengths and distances
- Minimize the bit representation of a numeric value based on its occurrence count



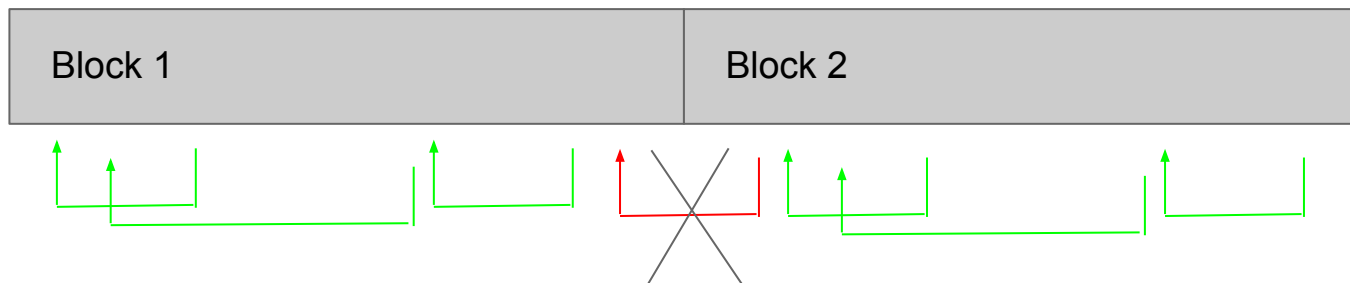
Symbol	Coding (bit values)
A	0
B	100
C	101
D	110
E	111

A few insights on Deflate

- Concepts of blocks.
- Compressors are free to use a single or multiple blocks
- Each block is preceded by 3 bit:
 - First bit:
 - 0: last block of the stream
 - 1: more blocks after this one
 - Second and third bit:
 - 00: Stored/uncompressed block of up to 65,635 bytes
 - 01: LZ77 + Static Huffman compressed tree (unlimited size)
 - 11: LZ77 + Dynamic Huffman compressed tree (unlimited size)
- A block can potentially reference symbols from preceding blocks ... and we don't want that!

What do we need ?

- To be able to access any part of the compressed stream without having to decompress the stream from its beginning
- 2 potential solutions:
 - Serialize regularly the state of the compressor, so that the decompressor can restart with it. But that's pretty large, at least 32 kB!
 - Or instruct the compressor to flush its sliding window at regular intervals, typically at block boundaries, or more exactly instruct it to **not** reference sequences of preceding blocks



What do we need ?

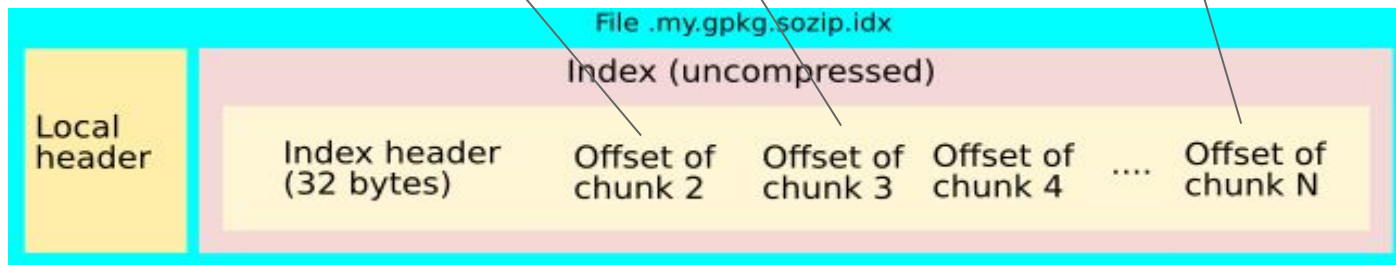
- Venerable ZLib library has a “full flush” mode that:
 - Resets the encoding dictionary
 - Align the compression stream with byte boundaries
 - Emits a 0-byte uncompressed block as a signature between 2 compressed blocks
 - Enables a decoder to start decoding the new block without knowing anything about the preceding block
- Technique used by the pigz (<https://zlib.net/pigz/>) Parallel GZip utility
- All compliant Deflate/ZIP readers can deal with that. They don't even realize that a “full flush” has been done !

SOZip ingredients

- Compressed stream, structured in chunks, each terminated by a ZLib “full flush”

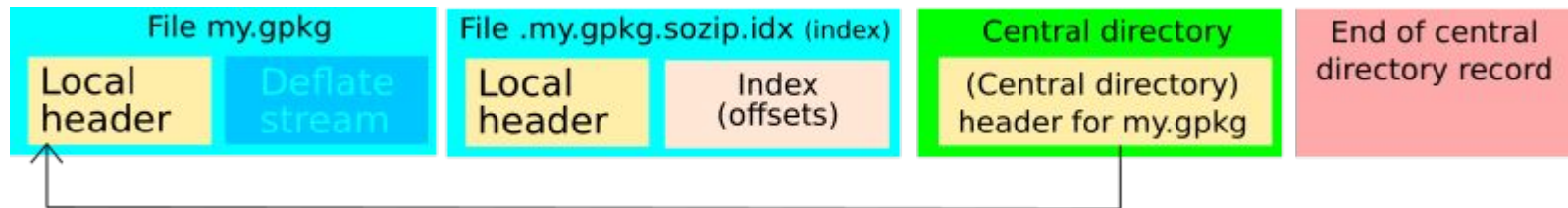


- A index file pointing every X bytes of uncompressed data to the offset of the beginning of the corresponding chunk:



Structure of a SOZip file

- Just a ZIP file:
 - “End of central directory record” marker
 - Central directory: index of all files, that point to the beginning of each file
 - And we find for each SOZip-enabled file in the archive:
 - The Deflate compressed stream (chunked) preceded by its “local” header
 - A **hidden** file that stores the index mapping uncompressed offsets to compressed ones



SOZip pros & cons

- Pros:
 - Make ZIP a workable format for random access
 - Multithreaded compression/decompression of independent chunks possible
 - For decompression, each chunk can be decompressed with a fast alternative, like libdeflate
 - Excellent backward compatibility: a data producer may deliver a SOZip enabled file with good confidence that nearly all existing ZIP readers can decompress it (at time of writing, we are not aware of ZIP readers that reject a SOZip enabled file)
- Cons:
 - Inherits the same limitations of the Zip format
 - Slightly degrade compression rate (dependent of the chunk size). Typically 2% with a 32 KB chunk size

GDAL implementation



- Available in GDAL 3.7 (released in May 2023)
- Existing /vsizip/ virtual file system handler enhanced to:
 - Generate SOZip-enabled archives (for files sufficiently big)
 - Can use multi-threading (like pigz) to compress files
 - Detect hidden SOZip index and use it to provide very fast random reading
- CPLAddFileInZip(): compress a file and add it to a new or existing ZIP file, and enable the SOZip optimization when relevant.
- VSIGetFileMetadata("/vsizip//path/to/my.zip/filename/inside", "ZIP") to get information if a SOZip index is available for that file.

GDAL implementation



- GeoPackage and Shapefile driver can directly generate SOZip enabled (.gpkg.zip / .shp.zip extensions):
⇒ `ogr2ogr my.gpkg.zip my.gpkg`
- New “sozip” command line utility:
 - List the content of a ZIP file and check if files in it are SOZip-optimized: “sozip -l my.zip”
 - Validate a SOZip file: “sozip --validate my.zip”
 - Create a SOZip file: “sozip my.zip my.gpkg meta.html”
 - Convert a regular ZIP to SOZip:
“sozip --convert-from=in.zip out.zip”



Python implementation

- <https://github.com/sozip/sozipfile> : fork of core “zipfile” Python module
- ⇒ “pip install sozipfile”
- Fully API compatible with “zipfile”
- Create a SOZip enabled file in a ZIP:

```
import sozipfile.sozipfile as zipfile
with zipfile.ZipFile('my.zip', 'w',
                    compression=zipfile.ZIP_DEFLATED,
                    chunk_size=zipfile.SOZIP_DEFAULT_CHUNK_SIZE) as myzip:
    myzip.write('my.file')
```

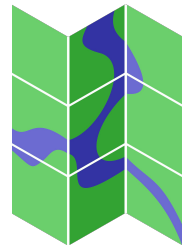
- Check if a file is SOZip enabled:


```
import sozipfile.sozipfile as zipfile
with zipfile.ZipFile('my.zip', 'r') as myzip:
    if myzip.getinfo('my.gpkg').is_sozip_optimized(myzip):
        print('SOZip optimized!')
```

Indirect implementations

- MapServer webmapping server:
 - MapServer 8.2 + GDAL 3.7 will produce SOZip enabled files when using a ZIP output format, such as:

```
OUTPUTFORMAT
  NAME "OGRGPKGZIP"
  DRIVER "OGR/GPKG"
  MIMETYPE "application/zip; driver=ogr/gpkg"
  FORMATOPTION "STORAGE=memory"
  FORMATOPTION "FORM=zip"
  FORMATOPTION "FILENAME=result.gpkg.zip"
END
```



-  with GDAL 3.7 automatically benefits from SOZip: drop a SOZip compressed GeoPackage, FlatGeoBuf, Shapefile, etc. And it is instantly opened and usable in a fully fluent way!

Benchmarking

- ZIP generation:

Timing	Action
6.1 s	Multithreaded (12 vCPUs) generation of 489 MB SOZip-enabled file from a 1.6 GB uncompressed GeoPackage file with <code>sozip nz-building-outlines.gpkg.zip nz-building-outlines.gpkg</code>
36 s	Single threaded compression of same file to 480 MB with regular <code>zip</code> utility with <code>zip nz-building-outlines-regular.gpkg.zip nz-building-outlines.gpkg</code>

- Bulk reading: Multithreaded extraction (4 vCPUs) of 3.2 million features with Arrow Array interface

Timing	Action
1.2 s	from SOZip-compressed GeoPackage file with <code>bench_ogr_batch nz-building-outlines.gpkg.zip</code>
0.7 s	from uncompressed GeoPackage file with <code>bench_ogr_batch nz-building-outlines.gpkg</code>

Benchmarking

- Subsetting: Extraction of 66,377 features with a spatial filter

Timing	Action
1.2 s	from SOZip-compressed GeoPackage file with <code>ogr2ogr out.gpkg nz-building-outlines.gpkg.zip -spat 1740000 5910000 1750000 5920000</code>
1.1 s	from uncompressed GeoPackage file with <code>ogr2ogr out.gpkg nz-building-outlines.gpkg -spat 1740000 5910000 1750000 5920000</code>

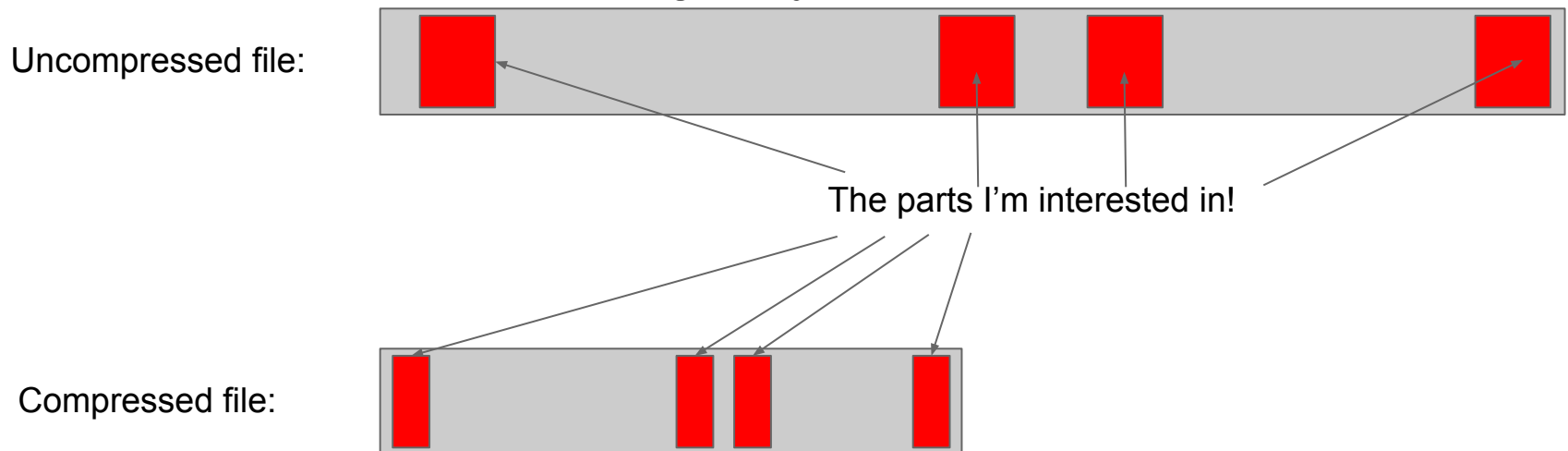
- Extraction of one feature from its identifier:

Timing	Action
45 ms	from SOZip-compressed GeoPackage file with <code>ogr2ogr out.gpkg nz-building-outlines.gpkg.zip -fid 1000000</code>
44 ms	from uncompressed GeoPackage file with <code>ogr2ogr out.gpkg nz-building-outlines.gpkg -fid 1000000</code>

**And now the question you all
wonder about....**

Is it cloud optimized/friendly... ?

- Yes and no
- SOZip by itself doesn't make a non cloud-optimized uncompressed format magically optimized



- But if a uncompressed format is cloud-optimized, it will remain cloud-optimized after ZIP compression. ⇒ /vsicurl/ + /vsizip/ can become cloud optimized

What remains to be done?

- Mostly use it!
 - We encourage data producers and distributors to adopt it
 - Existing readers will not be affected
 - SOZip aware readers will benefit from it
- ⇒ Same idea as the COG (Cloud Optimized GeoTIFF) or COPC (Cloud Optimized Point Cloud) formats
- More implementation in other languages: Javascript, etc. ?



Questions?

Credits to Safe Software for funding this effort

Links:

<http://sozip.org/>

Contact: even.rouault@spatialys.com





Spare slides

Related works

- <https://github.com/minio/zipindex>: a size optimized representation of a zip file directory to allow decompressing the file without reading the zip file index.
- <https://github.com/linz/cotar> (Cloud-optimized TAR): similar but for .tar format
- <https://github.com/vasi/lzopfs#gzip>: FUSE filesystem allows you to view a compressed file as if it was uncompressed, including random access operations. Uses the space consuming technique of storing the compressor state at regular intervals