Blosc2: A fast, compressed and persistent data store library

Francesc Alted - @FrancescAlted
The Blosc Development Team
CEO ironArray.io [ironArray]

LEAPS Innov WP7 (data reduction and compression) meeting
October 11th 2021
Back in the 1940’s, Claude Shannon invented a way to measure the information content of a message and called it information entropy:

$$H(s) = - \sum_{i=1}^{n} p_i \log_2(p_i)$$

In theory, you cannot compress a dataset beyond that entropy.

However, Shannon did not take into account that symbol ordering (and not only probability of occurrence) is important when finding ways to express messages in less space than such information entropy.
Breaking Entropy (II)

Blosc comes with so-called filters that are about re-ordering data before the encoding stage. One example is the shuffle filter:

This typically allows codecs to go beyond information entropy limits.

BTW, Blosc2 has optimized versions of the shuffle filter for Intel (SSE2, AVX2), ARM (NEON) and PowerPC (ALTIVEC, thanks to a ESRF grant).
What is Blosc?

✓ Sending data from CPU to memory (and back) faster than `memcpy()`.

✓ Split in blocks for better cache use: divide and conquer.

✓ It can use different filters (e.g. shuffle, bitsuffle) and codecs (e.g. LZ4, Zlib, Zstd, BloscLZ).
Leveraging Blosc the ‘right way’

- Blocks should be decompressed and operated in private caches for best performance.
- The need for data to fit in private caches is to avoid contention in Blosc multithreading.
- If possible, use all the data before it leaves caches.
Compression and decompression speed

https://www.blosc.org/posts/breaking-memory-walls/
What is Blosc2?

- Blosc2 is the next generation of Blosc(1).
- Blosc2 adds 63-bit containers (super-chunks) that expand over the existing 31-bit containers (chunks) in Blosc1.
- Metalayers for adding info for apps and users.
Blosc2: New features

Filter Pipeline | Serialization
Parallel I/O | Format

Pluggable Codecs & Filters
Blosc2: New features

<table>
<thead>
<tr>
<th>Filter Pipeline</th>
<th>Serialization Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel I/O</td>
<td>Pluggable Codecs &amp; Filters</td>
</tr>
</tbody>
</table>
Filter pipeline

Compression process

Decompression process

Filters pipeline

Filters:
- BLOSC_SHUFFLE          1
- BLOSC_NDCELL           32
- urfilter1                         160
Blosc2: New features

- Filter Pipeline
- Parallel I/O
- Serialization Format
- Pluggable Codecs & Filters
Frames can live either on disk or in memory
Frame specification is very simple

- Fully documented in **less than 700 lines of text:**
  
  ```
  > wc -l README_*_FORMAT.rst
  278 README_CFRAME_FORMAT.rst
  283 README_CHUNK_FORMAT.rst
  76 README_SFRAME_FORMAT.rst
  637 total
  ```

- One of the reasons is that it rests on the shoulders of MessagePack ([https://msgpack.org](https://msgpack.org)), an efficient binary serialization format.
- Simplicity is important in terms of portability, and specially, safety.
Blosc2: New features

Filter Pipeline

Parallel I/O

Serialization
Format

Pluggable Codecs
& Filters
Filters and codecs work in parallel

Compression process

- **Thread 1**
  - src1

- **Thread 2**
  - src2

- **Thread 3**
  - src3
Filters and codecs work in parallel

Compression process

Thread 1: src1 → tmp1
Thread 2: src2 → tmp1
Thread 3: src3 → tmp1
Filters and codecs work in parallel

Compression process

Thread 1
- src1 → tmp1 → tmp2

Thread 2
- src2 → tmp1 → tmp2

Thread 3
- src3 → tmp1 → tmp2
Filters and codecs work in parallel

Compression process

Thread 1
src1 → tmp1 → tmp2 → c_src1

Thread 2
src2 → tmp1 → tmp2 → c_src2

Thread 3
src3 → tmp1 → tmp2 → c_src3

Frame either on disk or in memory
Filters and codecs work in parallel

Decompression process

Thread 1

Thread 2

Thread 3

Frame either on disk or in memory

\begin{align*}
c_{\text{src1}} & \\
c_{\text{src2}} & \\
c_{\text{src3}} &
\end{align*}
Filters and codecs work in parallel

Decompression process

Thread 1

Thread 2

Thread 3
Filters and codecs work in parallel

Decompression process

Thread 1
- tmp1
- tmp2
- c_src1

Thread 2
- tmp1
- tmp2
- c_src2

Thread 3
- tmp1
- tmp2
- c_src3

Frame either on disk or in memory
Filters and codecs work in parallel

Decompression process

Thread 1

src1 → tmp1 ← tmp2 → c_src1

Thread 2

src2 → tmp1 ← tmp2 → c_src2

Thread 3

src3 → tmp1 ← tmp2 → c_src3

Frame either on disk or in memory
**Block masks and parallel I/O**

<table>
<thead>
<tr>
<th>Block maskout</th>
<th>F</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>F</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
</tbody>
</table>

Specially effective when retrieving slices of multidim datasets.
Masked & parallel I/O in multidimensional datasets

Much more selective and faster queries!

**Caterva** (https://github.com/Blosc/caterva) and **ironArray** (https://ironarray.io)
Masked & parallel I/O in multidim datasets

Better performance in general (except for dimensions where retrieving a chunk is already optimal).

Blosc2: New features

- Filter Pipeline
- Parallel I/O
- Serialization Format
- Pluggable Codecs & Filters
Adaptability: plugins in local registry

Filters registry

<table>
<thead>
<tr>
<th>Filter</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOSC_SHUFFLE</td>
<td>1</td>
</tr>
<tr>
<td>BLOSC_BITSHUFFLE</td>
<td>2</td>
</tr>
<tr>
<td>BLOSC_DELTA</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>BLOSC_NDCELL</td>
<td>32</td>
</tr>
<tr>
<td>BLOSC_NDMEAN</td>
<td>33</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>urfilter1</td>
<td>160</td>
</tr>
<tr>
<td>urfilter2</td>
<td>161</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

User defined filter:
```c
int urfilter2(blosc2_filter *filter) {
    ...
}
```

To register locally:
```c
blosc2_register_filter(urfilter2)
```

Can be used now:
```c
cparams.filters[4] = 161;
```

And a similar procedure goes for codecs too!
Registering plugins in central registry

User plugin
Registering plugins in central registry

User plugin

GitHub
Pull Request

Blosc development team
Registering plugins in central registry

User plugin → GitHub → Pull Request → Blosc development team → Evaluation process → To global registry
Registering plugins in central registry

User plugin

GitHub
Pull Request

Blosc development team

Evaluation process

To global registry

Specifications not fulfilled
Plugins in central registry are easy to deploy

Central registered plugins are **included** and **distributed** within the Blosc2 library, which can be installed using the Python wheels:

```
bash-3.2$ pip install blosc2 --no-cache-dir
Collecting blosc2
   Downloading blosc2-0.2.0-cp39-cp39-macosx_10_9_x86_64.whl (4.0 MB)
     | 4.0 MB 3.4 MB/s
Installing collected packages: blosc2
Successfully installed blosc2-0.2.0
```

Very convenient in making your filter/codec accessible for everybody
Other features for Blosc2

• **Safety/Security**: we are actively using the OSS-Fuzz service for uncovering programming errors in C-Blosc2.

• **Nice markup for documentation**: See [https://c-blosc2.readthedocs.io](https://c-blosc2.readthedocs.io)

• **Efficient support for special values**: repeated values can be represented with an efficient, simple and fast run-length encoding.

• **Preliminary Python wrapper for Blosc2**: [https://github.com/Blosc/python-blosc2](https://github.com/Blosc/python-blosc2)
Conclusion
Adapting compression to your needs

➔ Tackling compression includes a gazillion ways to do it, but basically:
  ➔ Get the maximum compression ratio
  ➔ Reduce the compression/decompression time to a maximum

➔ Blosc2 comes with a **rich set of codecs and filters** that users can easily try to find the one that better fits to their needs

➔ Blosc2 orchestrates these codecs and filters for:
  ➔ **Parallelization** via multithreading
  ➔ Reuse and sharing internal buffers for **optimal memory consumption**

The result is a highly efficient tool for **compressing your way**
Data is the most important part of your system

The Blosc development team is committed to the future of your data:

• Blosc2 has a very simple format, and hence is very portable and maintainable
• We have spent quite a lot of energy keeping it orderly and clean
• Last but not least, safety/security is paramount for us

**Proactivity** should be the primary mechanism of data integrity
<table>
<thead>
<tr>
<th>The Blosc Development Team</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aleix Alcacer</td>
</tr>
<tr>
<td>Oscar Guiñón</td>
</tr>
<tr>
<td>Marta Iborra</td>
</tr>
<tr>
<td>Alberto Sabater</td>
</tr>
<tr>
<td>Nathan Moinvaziri</td>
</tr>
<tr>
<td>Francesc Alted</td>
</tr>
</tbody>
</table>
Thanks to donors!

Without them, we could not have possibly put Blosc2 into production status: Blosc2 2.0.0 came out in June 2021; now at 2.0.4.
Enjoy data!

https://blosc.org/