Fast Serialization of Numpy Arrays with Bloscpack

Valentin Haenel

Freelance Consultant and Software Developer
`@esc___`

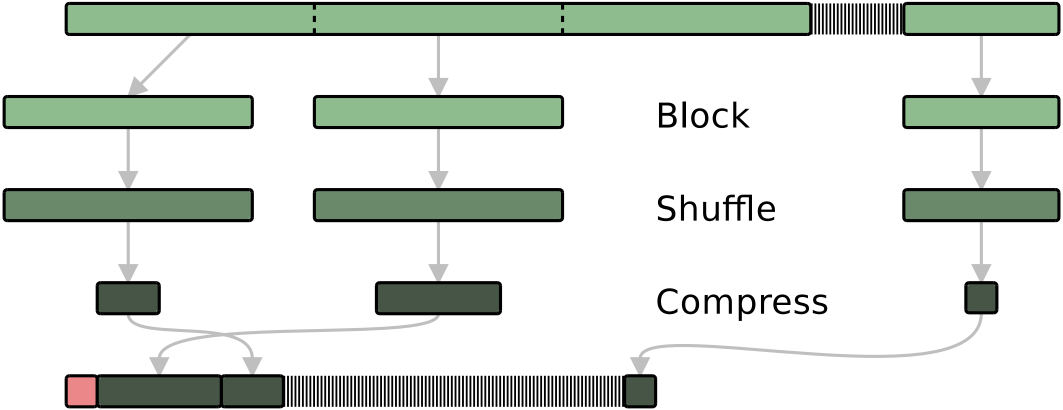
27 July 2014 - PyData Berlin

Version: 2014-PyDataBerlin `https://github.com/esc/PyDataBerlin2014-bloscpack`

Blosc

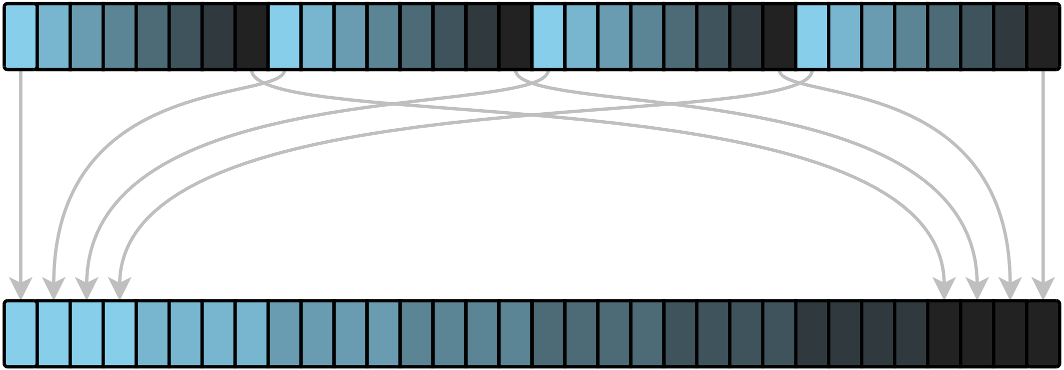
Blosc: A Fast Meta-Codec

- Blocking
- Shuffling
- Multithreaded
- Multi-codec



Shuffle Filter

- Reorder bytes by significance inside a block
- Potentially reduce Lempel-Ziv complexity of the data



Multi-Codec

- By default it uses **Blosclz** – derived from **Fastlz**
- Alternative codecs
  - **LZ4 / LZ4HC**
  - **Snappy**
  - **Zlib**

python-blosc: Bindings

- Python C-API bindings
- Accepts a pointer as int

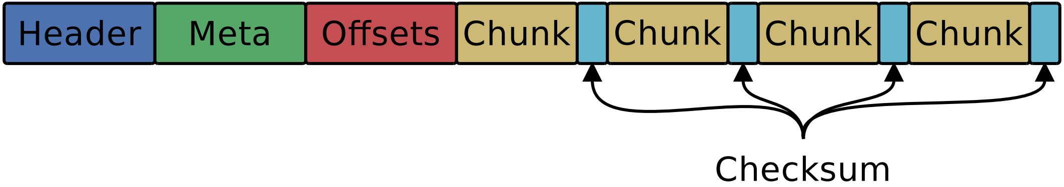
Bloscpack

Bloscpack

- Simple serialization format based on Blosc
- Command line interface
- Python API with support for Numpy arrays
- Aimed at developers / power-users

Features

- Chunked, compressed format
- Metadata (optional)
- Checksums (optional)
- Offsets, including pre-allocation for append (optional)



Use Cases

- Fast serialization
- Streaming
- On disk columnar storage
- Substrate on which to build high-level abstractions

Python API

- Inner loop compress/decompress implemented
- Concept: sinks and sources
  - `PlainSource` -> `CompressedSink`
  - `CompressedSource` -> `PlainSink`
- Supply appropriate source and sink
- Sources and sinks must obey an interface/contract
- Get easy Anything -> Anything
- E.g. Numpy -> {string, file, memory, network}

pack and unpack

```
def pack(source, sink,
         nchunks, chunk_size, last_chunk,
         metadata=None,
         blosc_args=None,
         bloscpack_args=None,
         metadata_args=None):
    pass

def unpack(source, sink):
    pass
```

Numpy Example

```python
import numpy as np
import bloscpack as bp

a = np.arange(1e7)

# pack with defaults
bp.pack_ndarray_file(a, 'a.blp')
```

Numpy Example

```
# pack with custom settings
bp.pack_ndarray_file(a, 'a.blp',
    chunk_size='20M',
    blosc_args=bp.BloscArgs(cname='lz4', clevel=9),
    bloscpack_args=bp.BloscpackArgs(offsets=False),
    )
```

Numpy Example

```
# unpack
b = bp.unpack_ndarray('a.blp')
```

Commandline Example

```
$ blpk compress --level 9 --codec lz4 --no-offsets data.dat
```

Extension Example

- Idea: how about S3 connectivity?
- Implement CompressedS3Sink and CompressedS3Source
- (These know nothing about Numpy)
- Result: ability to compress a Numpy array to an S3 bucket

Relationship to (Distributed) Analytics Engines

- Column-oriented, compressed, chunked storage
  - bcolz
  - Hustle
  - Parquet
  - RCFile / ORCFile
- Fast, partial loading from disk or network
- Reduced storage requirements
- But: need to chose the *right codec*^{TM}
- A Bloscpack file translates directly to a serialized column

Benchmarks

Background

- Builds on benchmarks presented at EuroScipy 2013
- Those used a laptop with SSD and SD storage
- Showed that Bloscpack can be outperform contenders

See also: Bloscpack: a compressed lightweight serialization format for numerical data

Experimental Setup

- Use Python 3.4
- Use some real-world datasets
- Benchmark new codecs available in Blosc
- Add PyTables to the mix
- Run it in the AWS cloud

Datasets

- **arange**
  - Integers
- **linspace**
  - floats
- **poisson**
  - more or less random numbers
- **neuronal**
  - Neural net spike time stamps
  - Kindly provided by Yuri Zaytsev
- **bitcoin**
  - Historical MtGOX trade data

Contenders

- PyTables
    - HDF5 interface
    - Supports Blosc and others
- NPY
    - Numpy plain serialization
- NPZ
    - Numpy compressed (using zip) serialization
- ZFile
    - Joblib's compressed (using zlib) **pickler** extension

NPY Flaw

- Prior to serialization, array is copied in memory with `tostring()`
- Fixed by Olivier Grisel to use `nditer` (#4077)
- Available in `v1.9.0b1`, which is what I used for the benchmarks

NPZ Flaw

- Create a temporary plain version (`/tmp`)
- Compresses into a Zip archive from there
- Due to issues with the ZipFile module

ZFile Flaw

- Does not support arrays larger than 2GB
- An `int32` is used somewhere for the size in the `zlib` module

Remaining Experimental Parameters

- Instance
  - c3.2xlarge
  - CPUs: 8
  - RAM: 15GB
- Dataset Sizes
  - 1MB
  - 10MB
  - 100MB
- Storage
  - EBS
  - Ephemeral

Measurements

- Writing to disk is tricky
- Measure with hot and cold FS cache
- Add disk `sync` to the timing
- Used a variant to the `timeit` utility.

Results

Let's look at the `arange` and `neuronal` datasets in the `small` and `large` configuration on `ebs` –> IPython notebooks

Aggregated Results

- Single plots can supply insights
- Need to aggregate for a big picture
- Award points to a codec/level combination
  - Slowest receives 1 point
  - Fastest receives max points
  - Compute with and without ratio
- Recommendation for a good general purpose codec
- See –> Ipython notebook `aggregate`

Conclusions – What did I Observe?

- Bloscpack vs. plain
  - In general it will not hurt to try Bloscpack
- Bloscpack vs. NPZ/ZFile
  - These formats don't scale well to large arrays
- Bloscpack vs. PyTables
  - Bloscpack is somewhat better at fast serialization
  - PyTables isn't the worst choice for long-term storage – but do use Blosc
- Blosclz vs. LZ4 vs. LZ4HC vs. Snappy vs. Zlib
  - Blosclz and LZ4 are the kings of fast compression
  - Snappy seems pretty average
  - Zlib can really benefit from Blosc acceleration and shuffle

Reproducibility

- Results contained in the talk sources repository
- Lists almost all the hashes and configurations
- All code open source
- All datasets additionally available from backup location on own infrastructure
- AMI available incl. instructions (soon to come / ask me)

TODO

- Find other ways to analyse the results
- Stabilize the format
- Release Python 3 support
- Support Bloscpack in Joblib
    - Speed gain
    - Mitigate 2GB issue

Getting In Touch

`http://blosc.org`

@*esc*___