Maximilian Knespel
Center for Information Services and High Performance Computing (ZIH)

# Ratarmount – A Tool for Mounting Large TAR Files

2020, June

# Why?

Why bother with mounting a TAR instead of extracting it?

- The file system might not support millions of very small files very well
- There might not be sufficient space for extracting
- The extracted uncompressed data might be multiple times larger
- You only need read-only access to a few files for previewing

# Why?

Why create a new tool?

Archivemount https://github.com/cybernoid/archivemount/commits/master exists for this but:

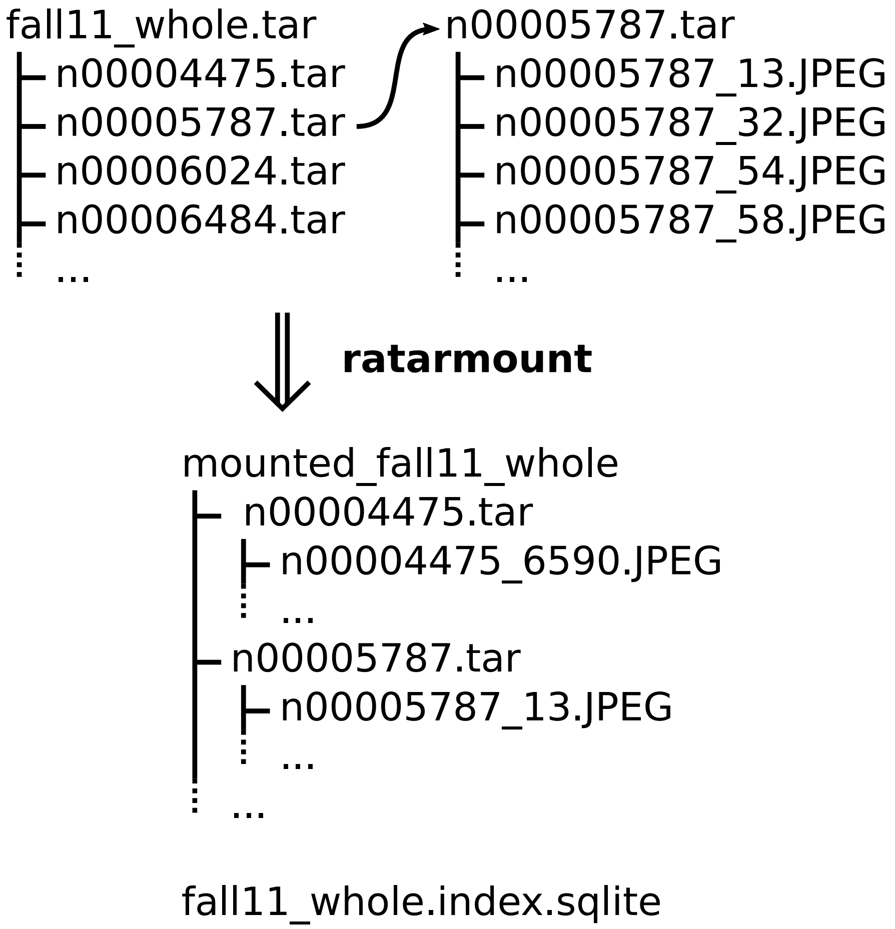
- Mounting can take hours
- No progress indicator during mounting
- Each subsequent mount will again take hours
- Accessing files can take more than a second
- Memory usage as long as it is mounted can be tens of gigabytes
- No integrated extended features like: recursive mounting, union mounting, support for updated files in the TAR, ...

# Ratarmount - RAndom Access TAR Mount

https://github.com/mxmlnkn/ratarmount

Main functions as required for mounting ImageNet:

- FUSE layer for easy access to files in TAR
- Find the offsets for each file to seek to
- Write out offsets and metadata to a file for reuse on subsequent mounting
- Keep memory usage low by streaming directly to and from the index file
- Support mounting TARs inside TARs recursively



fall11_whole.tar

n00004475.tar

n00005787.tar

n00006024.tar

n00006484.tar

...

n00005787.tar

n00005787_13.JPEG

n00005787_32.JPEG

n00005787_54.JPEG

n00005787_58.JPEG

...

**ratarmount**

mounted_fall11_whole

n00004475.tar

n00004475_6590.JPEG

...

n00005787.tar

n00005787_13.JPEG

...

...

fall11_whole.index.sqlite

# Ratarmount
## Additional features

Started out as a fairly simple ~300 lines python script.

These are some additional things initiated by reported issues on GitHub

- Support more obscure TAR features like sparse files and hard links
- Travis CI and automatic uploads to PyPI
- Support for bz2 and gzip compressed TAR files
- Mount not only tar.bz2 but also simple bz2 compressed files for uncompressed access
- Avoid memory leaks in tarfile module and by using SQLite for the index
- Work with incomplete TAR files
- Access older overwritten versions of a file analogous to tar --occurence=<n> ...
- Union mounting and bind mounting

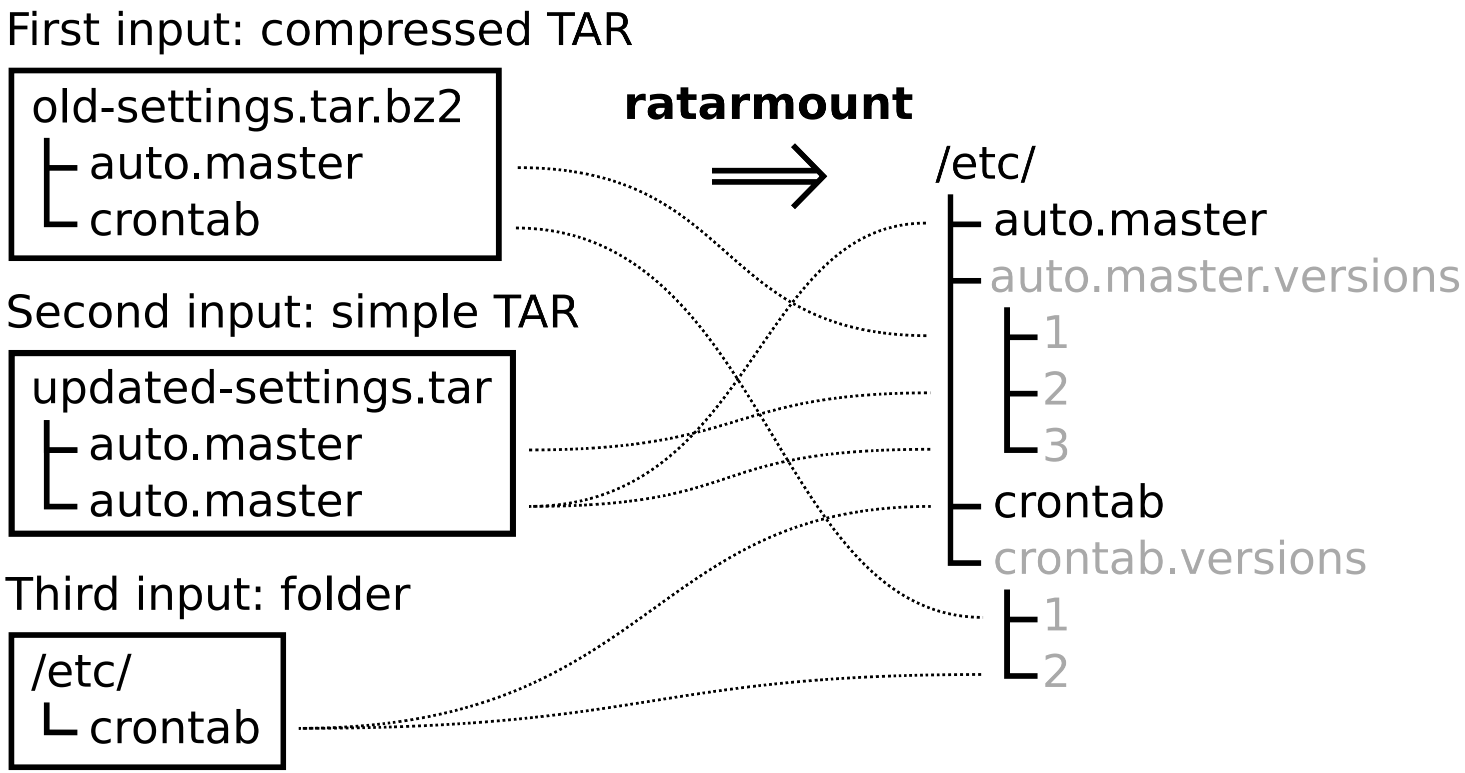
Now ~1700 lines, plus tests, plus the external indexed_bzip2 repository for bzip2 decompression.

# Ratarmount

## Union Mounting and file versions

Specify more then one input to get a merged view. An input folder **can** at the same time also be an output folder.

Access old versions of a <file> using the special hidden <file>.versions subfolder. The highest version number is the latest and the one shown by default.



First input: compressed TAR

old-settings.tar.bz2
┣ auto.master
┗ crontab

Second input: simple TAR

updated-settings.tar
┣ auto.master
┗ auto.master

Third input: folder

/etc/
┗ crontab

**ratarmount**

⇒

/etc/
┣ auto.master
┣ auto.master.versions
┃ ┣ 1
┃ ┣ 2
┃ ┗ 3
┣ crontab
┗ crontab.versions
  ┣ 1
  ┗ 2

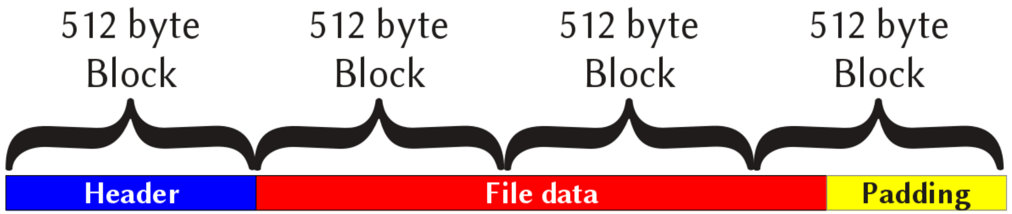
# How does it work?

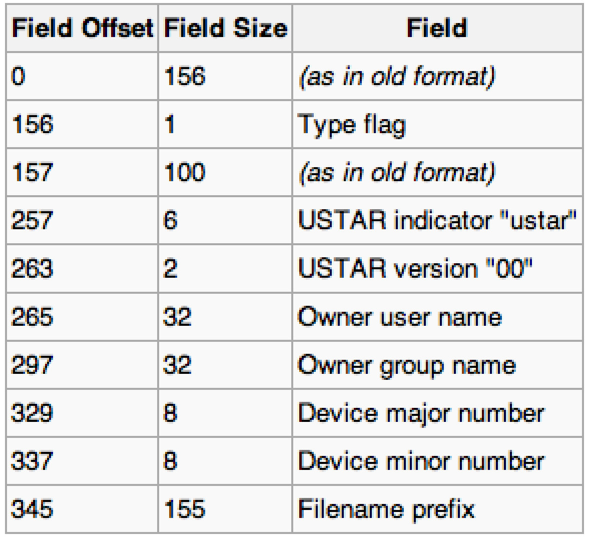
The tape archive format (TAR) is pretty simple. It basically contains the all files concatenated to each other aligned to 512B and interspersed with headers.

In order to seek to an arbitrary file, collect metadata and offsets:

| File Path | Seek Offset | File Size |
|---|---|---|
| n00004475.tar/<br>n00004475_6590.JPEG | 1024 | 78483 |
| n00004475.tar/<br>n00004475_15899.JPEG | 80384 | 94671 |

This is the bare minimum but other metadata like user name, id, … are also stored for FUSE.

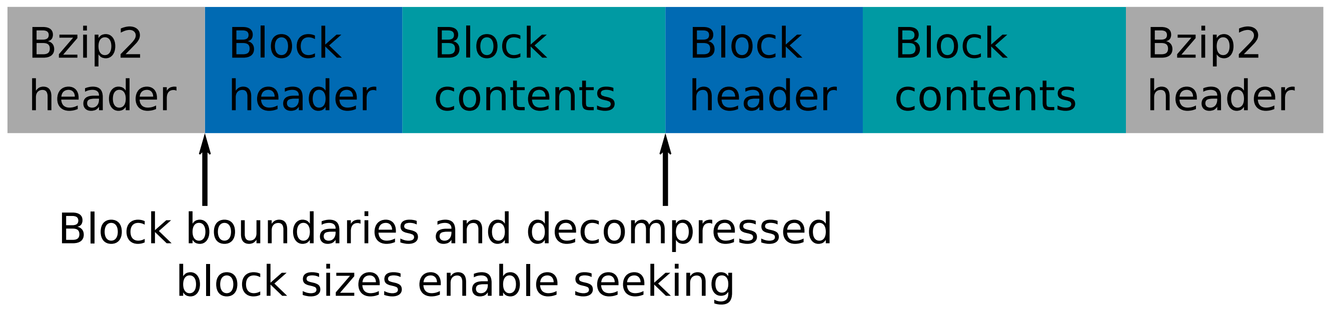




| Field Offset | Field Size | Field |
|---|---|---|
| 0 | 156 | *(as in old format)* |
| 156 | 1 | Type flag |
| 157 | 100 | *(as in old format)* |
| 257 | 6 | USTAR indicator "ustar" |
| 263 | 2 | USTAR version "00" |
| 265 | 32 | Owner user name |
| 297 | 32 | Owner group name |
| 329 | 8 | Device major number |
| 337 | 8 | Device minor number |
| 345 | 155 | Filename prefix |

# How does it work?

## Bzip2 Seeking



Collect the block header offsets with bit precision and the corresponding offsets in the decoded data. The latter is the accumulated sum of all decompressed block sizes before it.

I modified the 0BSD licensed Bzip2 decoder from https://github.com/landley/toybox

by Robert Landley to collect, export and import these offsets and added a Python interface in https://github.com/mxmlnkn/indexed_bzip2

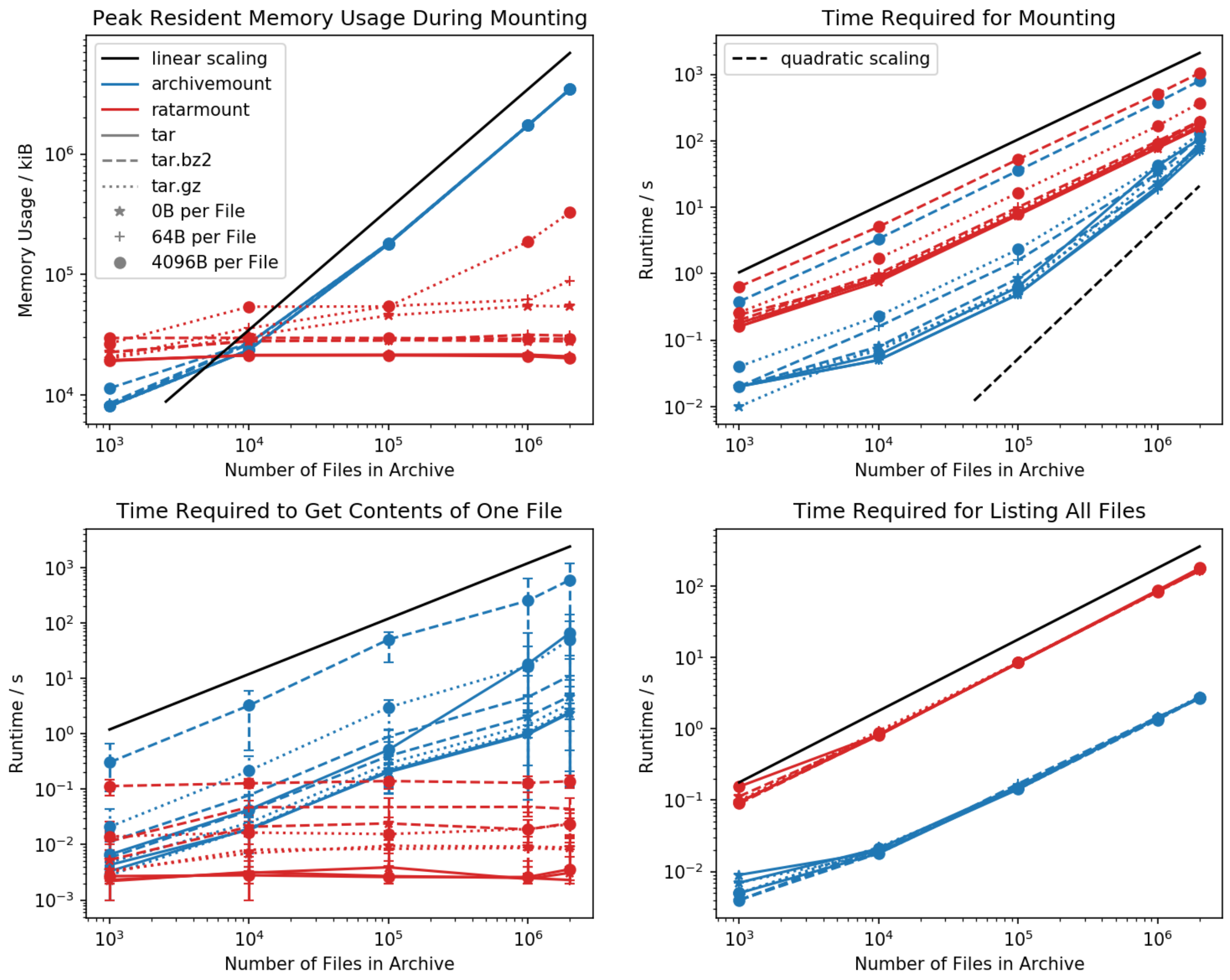
Gzip seeking done similarly with https://github.com/pauldmccarthy/indexed_gzip

by Paul McCarthy

# **Benchmarks**
## Comparison with archivemount

Storing and loading indexes alone is already a huge advantage. But, here are some other comparisons:

- Ratarmount memory usage is bounded except for the gzip backend.
- Mounting time unexpectedly is slower than archivemount at least for < 2M files. Scaling suggests an inversion.
- Accessing a file scales with the number of files when using archivemount!
- Tree traversal with the find command is for some reason 10x slower in ratarmount



# Benchmarks

## Index Serialization Backends

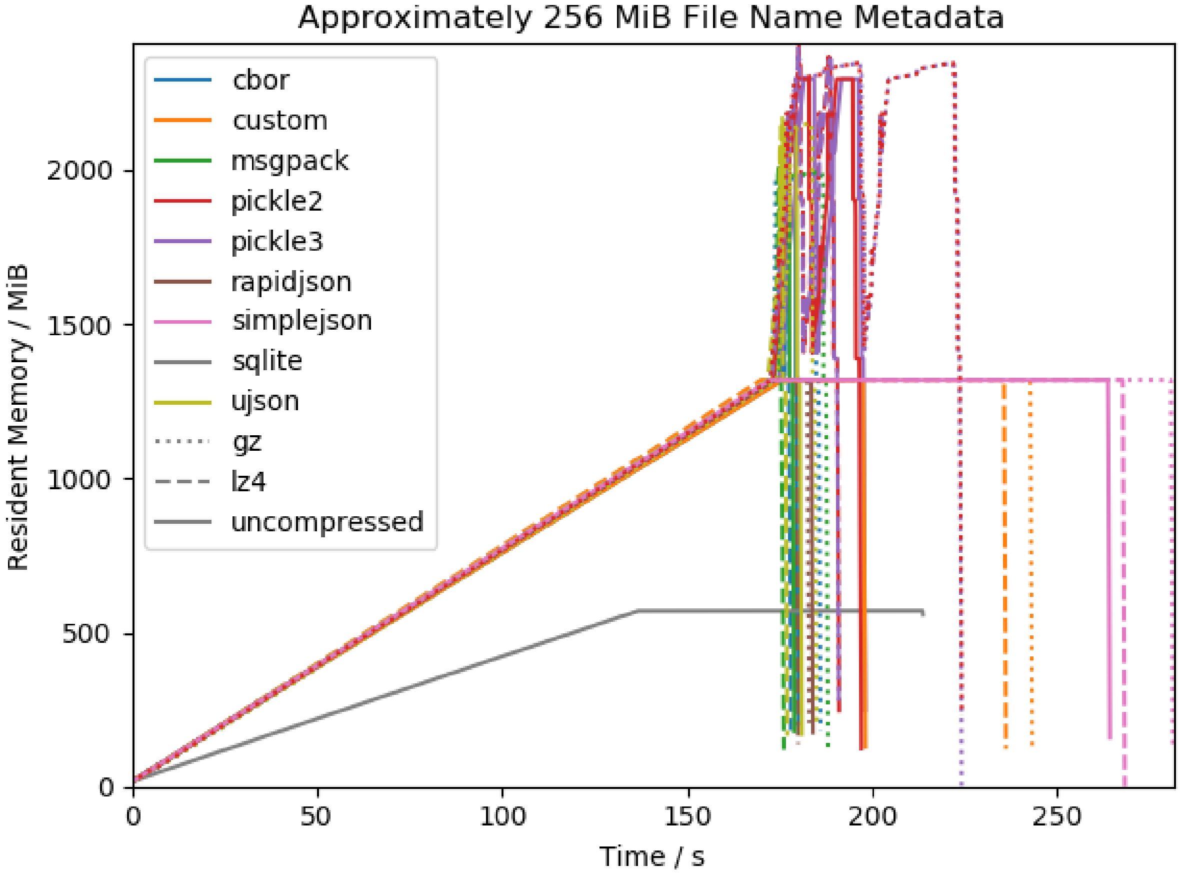
Periodically measure the resident memory size for different backends, which all save the same metadata.

Two phases:

- Metadata aggregation in memory
- Metadata dumping

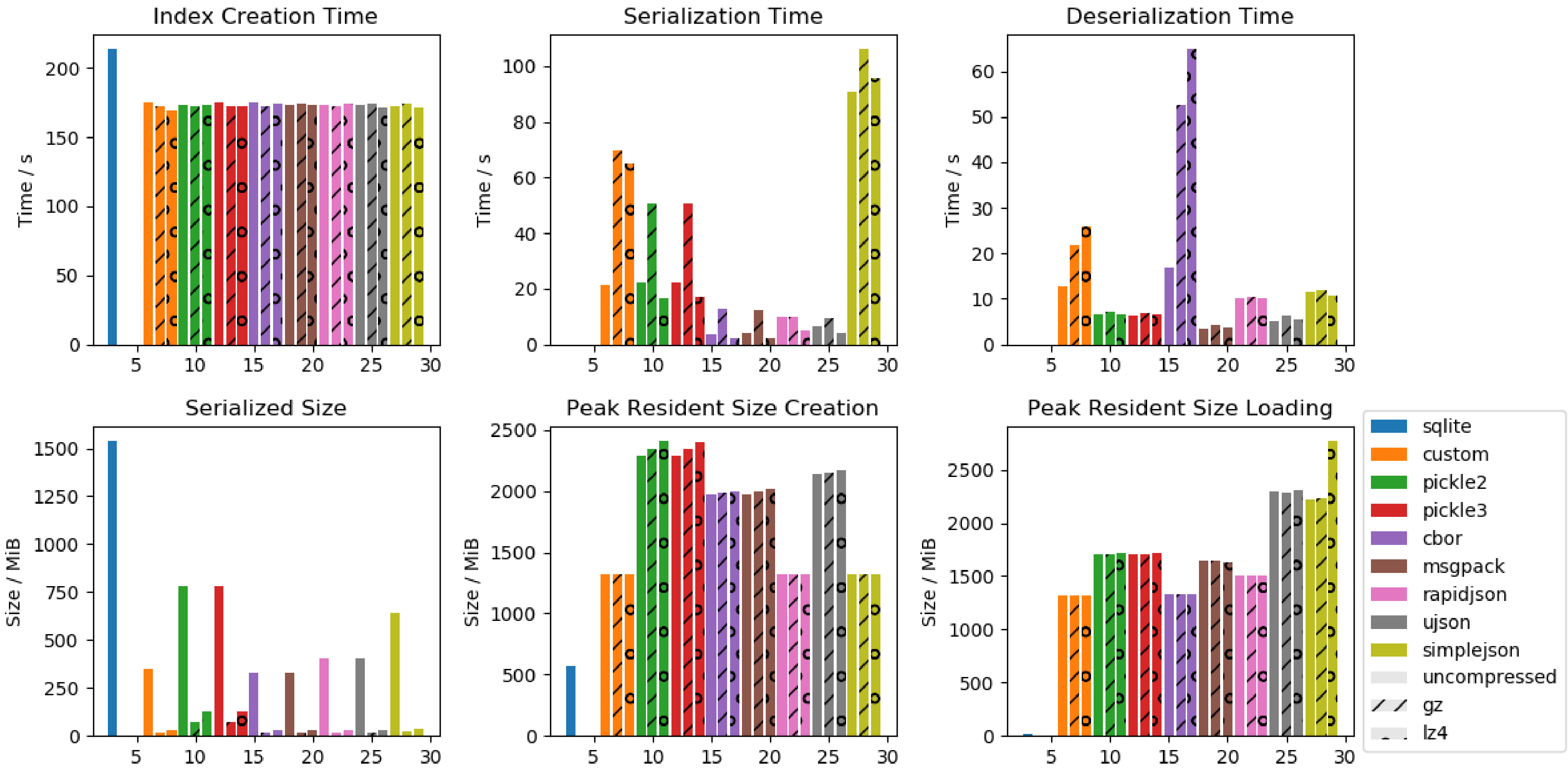
Results:

- Only SQLite has bounded memory usage
- Many backends double their memory usage during actually writing data



# Benchmarks

## Index Serialization Backends



# Future work

Ideas and ToDos:

- Optimize: sequential reading, bzip2 decoding, tree traversal with find
- Parallelize the Bzip2 decoder using one thread per bzip2 block and some kind of prediction
- Parallelize TAR indexing?
- Option for a writable overlay layer similar to Singularity has
- Limited write support, which would append updated and new files to the TAR?
- Add xz support? https://github.com/tomorrow-nf/tar-as-filesystem by Kyle Davidson and Tyler Morrow seems to be able to do this using a modified xz decoder.
- Support other systems than Linux, e.g., by using reFUSE a fork of fusepy