# USGS Collection 1 Reformatting tests and comparisons

Josh Sixsmith

December 14 2018

# 1 Introduction

The Level-1 Collection-1 data from USGS are distributed as compressed tarballs **.tar.gz* and will be used as the primary input for producing Geoscience Australia's Landsat Collection-3.
The purpose of the distribution format was to reduce the size of the data stream users were downloading whilst maintaining backwards compatibility for the individual TIFF files. As such, this report is not to highlight issues with the distribution format, instead it is to make Geoscience Australia aware of the limitations the original downloaded format will have on its production processing system.
The tarballs are compressed as a single stream using gunzip. This presents an issue in random access of the TIFF files, with the full tarball stream requiring decompression before allowing random access on a TIFF file.
Initial testing of the original downloaded files proved to be usable, but extremly slow to process. Unpacking the data is not suitable on the infrastructure we're using for processing, as it significantly increases the storage use in terms of both size and number of files.
The aim of this report is to test and compare different package structures whilst keeping as close as possible to the original format and structure of the files distributed by the USGS and provide GA with the information required to make an informed decision regarding the operational status of the USGS Collection-1 archive.

# 2 Method

The original tarballs will be unpacked and compressed using GDAL and repacked using the following container formats:

- *.tar
- *.tar.gz
- *.zip

The TIFF files will compressed using the deflate algorithm, comparing aggression levels 6 and 9, and a chunksize of (256, 256). A full matrix of combinations of chunksize and aggression levels won't be evaluated, as we aren't testing for extreme performance or extreme compression. Merely a good balance beteen the two, whilst still providing a file format that can be easily interpreted by a general user with the need for specialist tools.
The reformatted versions, **.tar* and **.tar.gz* have gone through the exact same image compression settings *[deflate, level=9, chunks=(256, 256)]*, and the text files with no compression.
The reformatted *tar.gz* was compressed with *level=0*, meaning we'll zip the tar file, but apply no compression. These files will contain additional metadata in the file headers specific to the gzip settings. Hence the file size is only slightly larger; 793.535 vs 793.656.

# 3 Results

## 3.1 Acquisition test (metadata loading)

The original **.tar.gz* files took on average 1 minute and 21 seconds to read the metadata and build an Acquisition Python Class that provides direct I/O access to an image file. I would go so far to say that this structure is useless for operational purposes on our current environment.
The reformatted **.tar* files can now be read in $< 1$ second, whereas the reformatted **.tar.gz* files took nearly 7 seconds. This indicates that there is a significant overhear purely from gzip being applied to the tar container, even though an aggression level of 0, i.e. no compression, was used on the container itself.

Table 1: Package size, Parsing time, and parsing time relative to original

| **Format Type** | **Package Size** | **Parsing Time** | **Time % of original** |
|---|---|---|---|
| Original *.tar.gz | 945MB | 81.12 seconds | 100% |
| Reformatted *.tar.gz | 794MB | 06.71 seconds | 8.27% |
| Reformatted *.tar | 794MB | 00.14 seconds | 0.17% |

## 3.2 Production processing test: wagl

28 Landsat 8 scenes were selected for use in testing a production run. The test was run at NCI Australia, using a normal Broadwell node (28 CPU's, 64GB). For an additional comparison, the imagery were compressed with deflate level 6. It is anticipated that there will be no significant overhead in reading data from deflate level 6 compared to deflate level 9.
The test was merely to see how long it takes to run wagl over the different versions of archive reformatting, and produce NBAR/NBART.
The different archive formats are:

- Original *tar.gz* from USGS
  - The imagery is compressed by gunzip, not GDAL, thus not allowing random access
  - Total storage size for the *tar.gz's*:
    - 27201244 bytes (26GB)
  - Walltime:
    - 03:06:17
  - CPU Time:
    - 17:56:32
  - Service Units:
    - 108.67
- Reformatted *tar.gz*

  - The imagery is compressed by GDAL with deflate 9, and the tarball itself is compressed with deflate 0
  - Total storage size for the 28 *tar.gz's*:
    * 22427016 bytes (22GB)
  - Walltime:
    * 01:07:18
  - CPU Time:
    * 13:05:24
  - Service Units:
    * 39.26

- Reformatted *tar.gz*
  - The imagery is compressed by GDAL with deflate 6, and the tarball itself is compressed with deflate 0
  - Total storage size for the *tar.gz's*:
    * 22448736 bytes (22GB)
  - Walltime:
    * 01:09:06
  - CPU Time:
    * 12:37:15
  - Service Units:
    * 40.31

- Reformatted *tar* with image chunk size (256, 256)
  - The imagery is compressed by GDAL with deflate 6 with a chunk size of (256, 256), and contained in a tar (no gzip over the top of the tar)
  - Total storage size for the *tar's*:
    * 22445280 bytes (22GB)
  - Walltime:
    * 00:23:51
  - CPU Time:
    * 09:15:11
  - Service Units:
    * 13.91

- Reformatted *zip* with image chunk size (256, 256)
  - The imagery is compressed by GDAL with deflate 6, and contained in a zip archive using the *ZIP_STORED* setting (no compression)
  - Total storage size for the *zip's*:
    * 22444908 bytes (22GB)

  - Walltime:
    * 00:38:25
  - CPU Time:
    * 11:54:10
  - Service Units:
    * 22.41

- Unpacked original *tar.gz*
  - Unpack the original *tar.gz* and nothing else
  - Total storage size:
    * 50983272 (49GB)
  - Walltime:
    * 00:39:08
  - CPU Time:
    * 11:01:48
  - Service Units:
    * 22.83

- Reformatted *tar* with image chunk size of (512, 512)
  - The imagery is compressed by GDAL with deflate 6 with a chunk size of (512, 512), and contained in a *.tar* (no gzip over the top of the tar)
  - Total storage size:
    * 22435732 bytes (22GB)
  - Walltime:
    * 00:33:11
  - CPU Time:
    * 09:16:58
  - Service Units:
    * 19.36

Table 2: Comparison of Storage Size, Walltime, CPU Time and Service Units

| **Format Type** | **Total Storage Size (bytes)** | **Walltime** | **CPU Time** | **Service Units** |
|---|---|---|---|---|
| Original *.tar.gz | 27,201,244 | 03:06:17 | 17:56:32 | 108.67 |
| Unpacked original *.tar.gz | 50,983,272 | 00:39:08 | 11:01:48 | 22.83 |
| Reformatted *.tar (chunk (256, 256)) | 22,445,280 | 00:23:51 | 09:15:11 | 13.91 |
| Reformatted *.tar.gz (deflate 6) | 22,448,736 | 01:09:06 | 12:37:15 | 40.31 |
| Reformatted *.tar.gz (deflate 9) | 22,427,016 | 01:07:18 | 13:05:24 | 39.26 |
| Reformatted *.zip | 22,444,908 | 00:38:25 | 11:54:10 | 22.41 |
| Reformatted *.tar (chunk (512, 512)) | 22,435,732 | 00:33:11 | 09:16:58 | 19.36 |

Table 3: Relative Comparison of Storage Size, Walltime, CPU Time and Service Units

| **Format Type** | **Total Storage Size (bytes)** | **Walltime** | **CPU Time** | **Service Units** |
|---|---|---|---|---|
| Original *.tar.gz | 100% | 100% | 100% | 100% |
| Unpacked original *.tar.gz | 187.43% | 21.00% | 61.48% | 21.00% |
| Reformatted *.tar (chunk (256, 256)) | 82.52% | 12.80% | 51.57% | 12.80% |
| Reformatted *.tar.gz (deflate 6) | 82.53% | 37.09% | 70.34% | 37.09% |
| Reformatted *.tar.gz (deflate 9) | 82.45% | 36.13% | 72.96% | 36.13% |
| Reformatted *.zip | 82.51% | 20.62% | 66.34% | 20.62% |
| Reformatted *.tar (chunk (512, 512)) | 82.48% | 17.81% | 51.74% | 17.81% |

# 4 Summary

Restructuring the '*.tar.gz*' files into a different container and compressing the imagery using GDAL decreased the filesize by approximately 16%, thus reducing our on-going storage costs. The reformatted "tar.gz" deflate level 6 and 9 didn't show any significant differences in compute costs. Thus I would recommend using deflate level 9, in order to gain the most from disk space savings.

The reformatted **.tar* files, in this test, appear to be the most effective at reducing the computational cost. The reformatted **.tar.gz* files appear to incur a significant overhead cost in reading the data, hence the increased computational time. The only difference between the **.tar* and the reformatted **.tar.gz* is the additional compression (deflate level 0) of the tar. The unpacked **.tar.gz* files, resulting in directories with uncompressed TIFF files, was significantly better in terms of computation cost than the original **.tar.gz* files. However, the tilling regime is row by row, resulting in significantly more reads than the tiled TIFF files. Additional work could be done on the wagl api side to improve reads by catering for non-native block reads, or more likely reading multiples of native block sizes. The other complexity is the increased disk space usage, by temporarily storing the full uncompressed archive, which would impede large scale production.

The comparison of chunk sizes (256, 256) vs (512, 512), saw some slight differences. The (512, 512) had a slightly smaller storage size, opposite to previous tests. Whereas the (256, 256) operated slightly faster. Keep in mind that this is a small sample for testing, and if required, repeated runs between these two setting could be processed and evaluated.

I would recommend that we reformat our Landsat Level-1 archive, from a compressed archive, to utilising compression within the TIFF itself, and storing in an archive/tarball with no compression i.e. **.tar* only. The TIFF's should be compressed using a chunk size of (256, 256) using deflate level 9.

I do recognise that the *tar* format doesn't support lengthy member names (unlike zip), as well as the fact that this format will differ to USGS's standard distribution format.

Some users might come into issue with the compressed GeoTIFF's, but most image processing packages these days have no problem in handling compressed GeoTIFF's (especially if using the deflate filer).

For Windows users, the *.tar* format is supported by tools such as WinZip, and 7Zip. From that perspective, any user making use of the standard USGS format will have no issues here, in effect it'll be 1 less operation, i.e no unzipping of the "tar.gz", then unpacking the ".tar".

The items addressed above will only come into effect for those users who download GA's flavour of USGS Level-1 data. Our internal systems will handle the files without issue.

The goal here is to reduce the cost of operational processing, in terms of KSU and time taken. Backlog processing could be achieved in quicker timeframes. As well as reducing the storage cost for the Level-1 data by / 16/

# A Parsing Metadata Performance Experiment (Python)

```
>>> orig_fname = 'LC08_L1TP_090091_20170819_20180203_01_T2.tar.gz'
>>> refmt_tar_fname = 'LC08_L1TP_090091_20170819_20180203_01_T2.tar'
>>> refmt_targz_fname = 'LC08_L1TP_090091_20170819_20180203_01_T2.tar.gz'
>>> def tictoc(fname):
...     st = datetime.datetime.now()
...     c = acquisitions(fname)
...     et = datetime.datetime.now()
...     print(et - st)
...
>>> tictoc(refmt_tar_fname)
0:00:00.142707
>>> tictoc(refmt_targz_fname)
0:00:06.714195
>>> tictoc(orig_fname)
0:01:21.124626
>>> orig_stat = os.stat(orig_fname)
>>> refmt_tar_stat = os.stat(refmt_tar_fname)
>>> refmt_targz_stat = os.stat(refmt_targz_fname)
>>> orig_stat.st_size / 1024 / 1024
945.9611234664917
>>> refmt_tar_stat.st_size / 1024 / 1024
793.53515625
>>> refmt_targz_stat.st_size / 1024 / 1024
793.6562995910645
>>> 1 - (refmt_tar_stat.st_size / orig_stat.st_size)
0.16113343713103556
>>> 1 - (refmt_targz_stat.st_size / orig_stat.st_size)
0.16100537336809728
```