# Compression Performance Evaluation Using Repack

July, 2005

## Introduction

The *hrepack* [3] and *h5repack* [4] tools are used to logically copy all the objects in an HDF4 or HDF5 file to a second file. These tools allow the new object to be written out with specified storage parameters, such as chunking or compression. Therefore, these tools can be used to write sample datasets using different options, e.g., to evaluate the effectiveness of data compression.

This document describes two utilities, *h5_compress_test.sh* and *h4_compress_test.sh*, that use the repack programs to re-write input files using SZIP and GZIP compression using all possible settings for the compression methods. The results can be used to evaluate the best settings for the sample data.

Essentially, the script runs *hrepack* (or *h5repack*), applying compression to all objects in the input file. It is important to understand that this process will compress whatever objects can be compressed, generally, all datasets with more than 1KB of numeric data. For a simple file with a few numeric datasets, most of the file will be compressed. For a file with many attributes, annotations, groups, or other objects, only some of the file will be compressed.

Note, too, that this test uses default chunking strategies. For HDF4, the chunking (or contiguous) layout of the input dataset is used for the output. For HDF5, if the input dataset is chunked, the output will be chunked the same way. If the input HDF5 dataset is not chunked, the output dataset will be chunked in one chunk.

Similarly, other storage options (e.g., the HDF5 shuffle filter, the HDF4 JPEG compression) are not applied in these scripts.

## Compression

These scripts apply GZIP and SZIP compression, repeating the repack with each possible parameter setting for the respective compression method.

GZIP has one parameter, “level” (Table 1). This controls the compression algorithm, with higher values running longer to produce greater compression (smaller output).

**Table 1. GZIP parameter (See [2]).**

| Parameter | Values | Description |
|---|---|---|
| Level | 1-9 | 1= less compression, 9 = more compression |

SZIP has several parameters, but only two are set by the user program (Table 2). (The other parameters are automatically set by the HDF5 library.) The coding scheme may be set to "Entropy Coding" or "Nearest Neighbor". Please refer to the SZIP documentation for an explanation of this parameter. The second parameter is a blocking factor, "Pixels per Block". Together, there are 32 different combinations that may be set for SZIP compression.

**Table 2. SZIP parameters set by HDF tools (See [1])**

| Parameter | Values | Description |
|---|---|---|
| Coding | Entropy Coding, Nearest Neighbor | See [citation[ |
| Pixels Per Block | Even number, from 2-32 | Blocking factor |

**What the Script Does**

The two scripts perform the same operations, one for HDF4 and the other for HDF5.

For each input file, the script calls repack with each combination of parameters for SZIP and GZIP ( a total of 40 runs per file ).

First, the input file is copied to the work area.

For each combination of SZIP parameters, repack is called, and the time to complete the repack is saved. The size of the output file is saved. Then repack is called again with the compressed file as input, to write an uncompressed file. The time to uncompress is saved.

This process is repeated for GZIP.

Summary tables are written in the work area.

**Using the Script**

The *perf_test.sh* is a Unix shell script. The usage is:

```
h5_compress_test.sh file.h5 [file2.h5 ...]
 or
h4_compress_test.sh file.hdf [file.hdf . . .]
```

Three parameters must be set in the script. (Table 3) These are set by editing the script file.

**Table 3. Parameters set by editing the script.**

| Variable | Value |
|---|---|
| verbose (optional) | "yes" extremely verbose output |
| WORKDIR (required) | Scratch area for writing files, and output results |
| HREPACK (required) | Path to the h5repack or hrepack script. |

## Output

For each input file, two output files are created, <infile>_tab1.txt and <infile>_tab2.txt. These files summarize the results for SZIP and GZIP compression respectively. For each parameter setting, the table gives the compressed size of the file, the amount of compression (original size – compressed size), the time to create the compressed file, and the time to create an uncompressed copy from the compressed file.

Table 4 shows example output for SZIP for a file called 'test_data_save.h5', originally 59,879,190 bytes. The first column is the parameter combinations ('2_EC' = Entropy Coding with 2 pixels per block, etc.) The second column is the compressed size of the file (e.g., for 2_EC, the output of repack was: 67,068,288). Note that the file actually grew for some cases. The third column is the amount of compression achieved. The fourth column is the time to write the whole file, and the fifth column is the time to rewrite the file with no compression.

**Table 4. example output for SZIP**

Results for SZIP /var/tmp/W/test_data_save.h5 (size 59879190)

| parms | comp | amount | ctime | uctime |
|---|---|---|---|---|
| 2_EC | 67068288 | -7189098 | 16.81 | 10.01 |
| 2_NN | 25076256 | 34802934 | 11.67 | 4.80 |
| 4_EC | 61940704 | -2061514 | 10.86 | 9.10 |
| 4_NN | 22414848 | 37464342 | 10.05 | 4.54 |
| 6_EC | 60704096 | -824906 | 10.11 | 5.66 |
| 6_NN | 21670784 | 38208406 | 9.67 | 4.42 |
| 8_EC | 59432160 | 447030 | 8.52 | 7.04 |
| 8_NN | 21106048 | 38773142 | 8.33 | 4.40 |
| 10_EC | 59444832 | 434358 | 9.37 | 5.23 |
| 10_NN | 20966112 | 38913078 | 7.80 | 6.85 |
| 12_EC | 59072128 | 807062 | 9.52 | 5.32 |
| 12_NN | 20817216 | 39061974 | 9.24 | 6.13 |
| 14_EC | 58850784 | 1028406 | 9.87 | 5.71 |
| 14_NN | 20743456 | 39135734 | 9.41 | 4.46 |
| 16_EC | 58677568 | 1201622 | 8.13 | 6.40 |
| 16_NN | 20905600 | 38973590 | 7.44 | 7.94 |
| 18_EC | 59093120 | 786070 | 10.35 | 7.88 |
| 18_NN | 20902784 | 38976406 | 8.74 | 6.86 |
| 20_EC | 59017248 | 861942 | 9.04 | 6.28 |
| 20_NN | 20800864 | 39078326 | 8.66 | 7.05 |
| 22_EC | 59335456 | 543734 | 8.95 | 5.09 |
| 22_NN | 20929920 | 38949270 | 8.67 | 7.30 |
| 24_EC | 59327264 | 551926 | 8.89 | 8.56 |
| 24_NN | 20912992 | 38966198 | 8.54 | 8.00 |
| 26_EC | 58538240 | 1340950 | 9.89 | 4.93 |
| 26_NN | 20686880 | 39192310 | 8.52 | 5.36 |
| 28_EC | 59404480 | 474710 | 8.85 | 5.15 |
| 28_NN | 20972320 | 38906870 | 8.63 | 7.72 |
| 30_EC | 59516928 | 362262 | 8.82 | 5.20 |
| 30_NN | 21032224 | 38846966 | 8.53 | 5.95 |
| 32_EC | 58643744 | 1235446 | 8.67 | 6.20 |
| 32_NN | 20821856 | 39057334 | 8.38 | 6.36 |

Table 5 shows example results for GZIP with the same file.

**Table 5. Example output for GZIP.**

```
Results for GZIP
parms comp  amount      ctime uctime
1     8752640     51126550    9.65  5.19
2     3062784     56816406    6.81  2.76
3     2928640     56950550    7.04  2.52
4     1383008     58496182    10.37 2.73
5     1368928     58510262    10.16 7.20
6     1359520     58519670    10.58 2.44
7     1364512     58514678    11.56 2.56
8     1364960     58514230    13.83 2.83
9     1364960     58514230    13.89 5.00
```

## What this Procedure Measures

It is important to understand that this process provides imprecise information about the effectiveness of the compression.

*Measures of Data Size*

This approach does not measure the compression ratio directly, but may be used to compare methods, and also to indirectly compute a compression ratio.

Table 6 presents the statistics for data size that will be collected. The precise meaning of these statistics depends on the input file. For a file with exactly one dataset, the statistics are easy to interpret. For a file with many objects (e.g., a MODIS dataset), the global size of the file and the corresponding change in size can only be interpreted by understanding which objects in the file were, in fact, compressed.

**Table 6. Data size statistics**

| Measured and Computed Statistic | Interpretation |
|---|---|
| 1. Size of file (no compression)[*] | The size of the *whole file*, including metadata, annotations (HDF4), attributes, and datasets that are not compressible. |
| 2. Size of file (compressed ) | The size of the whole file, with one or more objects compressed. |
| 3. (2) – (1) | This is the amount of compression (reduction in bytes) on one or more objects. Subtraction removes the (in principle) identical contributions for metadata and non-compressed objects. |

[*] Assuming the original file was not compressed. The script works with any input file, although the results are difficult to interpret if some of the objects in the input file are compressed or only partly written.

The amount of compression (3) for different methods should be directly comparable. For example, if SZIP with certain settings reduces the file by 30,000 bytes, and GZIP reduces it by 20,000 bytes, SZIP achieved better compression.

If the size of the compressed objects is known (e.g., there is one dataset, and we know the amount of uncompressed data in it), we can manually calculate the compression ratio from the amount of compression. For a dataset with uncompressed size *S*, the compression ratio would be *S – (3) / S*.

For HDF5, the stored size of a dataset can be read from *h5dump* output, although it might be more convenient to write a special program to extract this than to write a parser for *h5dump* output. A special program would be required to extract this information from HDF4.

Note that this value could deceptive if chunking is used. Compression is applied per chunk, so there is actually a distribution of compression ratios for the whole dataset. We can only measure the total size summed across all chunks.

Care must be taken that the repack does not change other storage parameters that may affect the size of the file. This can happen due to default behavior of the utilities. In particular, we must take care that the chunking is that same for all cases, and that fill values are handled the same way.

*Measures of compression and decompression time*

As in the case of the size, the time measurements are imprecise. The times include both I/O and compression/decompression time, as well as the overhead open, close, etc..

**Table 7. Compression/decompression time statistics**

| Measured and computed statisitics | Interpretation |
|---|---|
| 1. Time to repack uncompressed to uncompressed | Time to copy all the objects, with no compression. Includes open, close, other overhead, and the time to read and write all the objects in the file. |
| 2. Time to repack uncompressed file to compressed | Time to read uncompressed, and write compressed |
| 3. Time to repack compressed to uncompressed | Time to read compressed and write uncompressed |
| 4. (2) – (1) | The added (or reduced) time to compress and write the data |
| 5. (3) – (1) | The added (or reduced) time to decompress and write the data |

Table 7 shows the time measurements that could be collected. The current implementation collects (2) and (3). As in the case of the size measurements, interpreting these numbers depends on the input dataset.

The total time for compression (2) or decompression (3) can be directly compared for different settings and compression methods (assuming all else is equal).

In the event that we know the amount of data in the uncompressed and compressed cases, it may be possible to subtract out the data transfer times. However, the combined time to compress and write or read and decompress is probably the statistic of interest for most purposes.

**Discussion**

These scripts provide a simple way to obtain a coarse estimate of what compression method and settings work best for a particular dataset. Since the scripts are used with whole files, they are especially useful for analyzing the possible effectiveness of compression as a post-processing step for existing datasets.

The scripts may also be used as a template that can be customized, e.g., to test chunking strategies.

As discussed above, these scripts cannot give a precise measure of the compression ratio, or of the contribution of the compression to the overall time. Accurate measurement of these requires detailed knowledge of the contents of the file and which data are actually compressed. This is difficult or impossible to determine in a simple script.

It should be noted that in addition to the compression methods and parameters, the performance of compression depends on the structure of the data (especially the chunk size), the data type (e.g., integer versus floats), and the statistical properties of the data.

**References**

1. "Szip Compression in HDF Products", http://hdf.ncsa.uiuc.edu/doc_resource/SZIP.

2. "Zlib", http://www.zlib.net/

3. "NCSA HDF Tools", http://hdf.ncsa.uiuc.edu/hdftools.html

4. "HDF5 Tools", http://hdf.ncsa.uiuc.edu/HDF5/doc/Tools.html#Tools-Repack