SZIP Performance Study

Robert E. McGrath
March 10, 2005

Goal

The overall goal of this project is to illustrate the effectiveness of SZIP compression with HDF4 and HDF5. For comparison, GZIP (deflate) compression will also be used.

The basic measurements are compression time, decompression time, and compression ratio. Variables that may affect these include the data elements and the layout of the data (e.g., size and shape of the array, or chunking strategy).

Ideally, the study will focus on data of interest to NASA, either real NASA data, or data that is representative of NASA datasets.

This study is designed to be implemented as a summer project for a student. The task is separated into two tasks. Part 1 is a simple and flexible, but coarse measurement. The second is more precise study of specific synthetic datasets.

Part 1: Simple repack tests

The *hrepack* (HDF4) and *h5repack* (HDF5) tools can be used to logically copy the objects from one HDF file to another, optionally adding compression. These tools can be used to compare compression methods by repacking the same input file with different options for compression, chunking, and so on. One advantage of this method is that any input file can be used, so it can be replicated by users with their own data.

The procedure is:

1. time repack –I testfile.hdf –o testfile-none.hdf –f <no-compression> -l <chunking settings>
2. time repack –I testfile.hdf –o testfile-sz.hdf –f <szip settings> -l <chunking settings>
3. time repack –I testfile.hdf –o testfile-gz.hdf –f <gzip settings> -l <chunking settings>
4. compare size of the output files.

Decompression time can be measured by using repack to uncompress the compressed files.

This test can give a coarse-grained measure of compression ratio and time. The time includes the time to copy all the objects, the differences indicate the time for the compression methods. The size of the files include all the objects and metadata, most of which may be unaffected by the compression. If the input file has many objects in it, the test may compress selected objects, so the amount of compression can be more easily inferred.

This test can be encapsulated in a Unix shell script, which can potentially be used with many different input files, and any of the supported compression methods. The script should collect data and produce a text report suitable for further analysis. This script could be used by others to make their own measurements.

Technical Note: 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 1. 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.

Table 1 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.

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 2. 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 2 shows the time measurements that will be collected. As in the case of the size measurements, interpreting these numbers depends on the input dataset.

The added time for compression (4) or decompression (5) 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.

Part 2: Measure synthetic datasets

The first method above has three important limitations. First, the time measurement includes the execution of the whole repack tool, including a full copy of the whole file. The compression time may be a relatively small fraction of that total, and therefore difficult to measure.

Second, the repack utility represents only one access pattern, which is not representative of the performance for other patterns of read and write. In particular, the performance of a hyperslab selection cannot be determined from the repack test.

And third, the size of the whole file includes all objects and metadata, so the compression ratio for the actual compressed data may be difficult to determine precisely. In a simple case with only one dataset in the file, or only one that is compressed, there is no problem. But if there are multiple datasets that are compressed, the results need to be evaluated carefully.

To address these limitations, we can write a simple test program that reads synthetic datasets, measures the compression and decompression times more precisely, and measures the compression ratios more precisely. It is likely that this program will be less flexible than the repack approach.

To control for the effect of the data, we will extract sample datasets from real files (e.g., MODIS), and write them to a standard HDF4 and HDF5 file layout.

The test program will perform the following steps:

1. read the data file into memory
2. For each setting in { NONE, SZIP, GZIP, …}
    a. create an HDF4 (HDF5) file
    b. write one dataset using the settings
    c. close
3. For each setting in { NONE, SZIP, GZIP, …}
    a. reopen
    b. read the datasets
    c. close

The time for the write and read will be measured for each case. We know exactly what data is written, so this is a very precise measurement of the compression algorithm.

The compression ratio can be read from the files. For HDF5 we can get the storage size of the dataset from the *h5dump* utility. In principle, this information can be retrieved from the HDF4 file, although we may need to make a special utility.

Potentially, this program could implement different access patterns, e.g., to read the data by hyperslabs.

While this program is not difficult to write for any one input case, it is very difficult to make a program that can handle arbitrary input data. The program needs to know the names or identities of the datasets that should be compressed, their datatype, and data space. If chunking and/or access by hyperslab is implemented, the algorithm must adapt to the dimensionality and size of the object. And so on. It is difficult to write code that can do this for any arbitrary HDF file.

Therefore, this program will be a limited framework for benchmarking compression in HDF. Of course, users could take our code and modify it to do their own data.

Summary

These two tests will give us the ability to obtain coarse-grained evaluation of a variety of data and compression settings, and a fine-grained measurement of a more limited case.

We should be able to construct this in a way that users can run the tests on their own systems, and adapt to their own data relatively easily.

The overall deliverables will be:

1. a technical note (approximately 10-20 pages)
2. a suite of tools that may be adapted by users (details TBD)

Two questions remain open in this note:

1. what data to use
2. what platforms to test

Clearly, performance results can vary widely from platform to platform. Ideally, these test should be run on several platforms, and the results compared. Testing on multiple platforms increases the complexity and time required for the analysis and reporting. We should do as many platforms as possible, in the time available.

Management Use Only: Estimated Effort

The two approaches above are based on my experience in the last year. I have scripts that can be extended to implement #1. I have written programs to do part of #2, so I know how to do it.

It is important to realize that the software development is trivial beside the time it takes to run, analyze, and re-run the tests, and then write up the results.

Based on my experience, I would estimate that a student with knowledge of Unix scripts could implement the script for #1 in a month or so, and run a simple case in another week. The analysis could take several weeks.

Developing the code for #2 would be about the same, assuming they start from an example.

Overall, this is 10-12 weeks work for a student. Ideally, I can do some of the coding in advance, there would be more time for collecting data an analysis.

Part 1:

Coding:	4 weeks
Run:	1 week
Analysis:	4 weeks

Part 2:

Coding:	4 weeks
Run:	1 week
Analysis:	4 weeks

Note: for each additional platform, only minor coding is needed (to fix portability bugs!), the run time is the same (and can overlap with other activities). The analysis time will increase, because there are multiple comparisons to make. All told, I would add 2-4 weeks for each additional platform, mostly in the analysis activities.