# Chunking and Compression Performance Tool Requirements

**Albert Cheng**

This describes the needs of a tool to measure the chunking and compression performance of HDF5 files.

Compression of large datasets may reduce the file sizes but mismatches between the layout of the file datasets and the access pattern of the applications using the data can result in poor performance or excessive use of machine resources. This tool will allow users to access their data files using different parameters such as chunking sizes, compression methods, access patterns, and chunk cache settings. The tool will provide performance statistics to help users to find the optimum parameters to create and access their data files.

## 1 Introduction

This describes the requirements of the Chunking and Compression Performance (CCP) tool. The users are expected to be familiar with the HDF5 software. The HDF5 Software Documentation is available at http://www.hdfgroup.org/HDF5/doc/index.html.

### 1.1 Motivation

Compression of large datasets in S-NPP, JPSS and other HDF5 data files may substantially reduce the file sizes; in turn reducing disk space usages and file download time. However, mismatches between the layout of the files' datasets, the HDF5 instance's cache settings, and the access pattern of applications using the data can sometimes result in poor performance or exhausting machine resources when running the application. Whether designed in advance or modified in response to encountered problems, applications can be tuned to optimize efficiency of data access, avoid unnecessary repeated decompression, and reduce the amount of memory used.

This tool will allow users to access their data files using different parameters such as chunking sizes, compression methods, access patterns, and chunk cache settings. The tool will provide performance statistics to help users to find the optimum parameters to create and access their data files.

## 2 Use cases of the CCP tool

The study of the performance of writing and reading HDF5 data files needs a tool to measure the I/O speeds of accessing the HDF5 data files, using different parameters such as chunking sizes, compression methods, access patterns, and chunk cache settings. We will show several use cases of how the CCP tool can be used to help users to find the optimum parameters to access HDF5 data files.

## 2.1 Achieving better performance on reading compressed files

A user has an HDF5 file with compressed datasets. The user may use the CCP tool with parameters such as access patterns and chunk cache settings, similar to his read application to access the file to get a baseline of performance. He then uses the tool with different sets of parameters to see how they may affect the performance comparing to the baseline performance, to find the optimum set of parameters for reading the file. He can apply the results to tune his read application to achieve the optimum performance. Figure 1 illustrates the overall process.

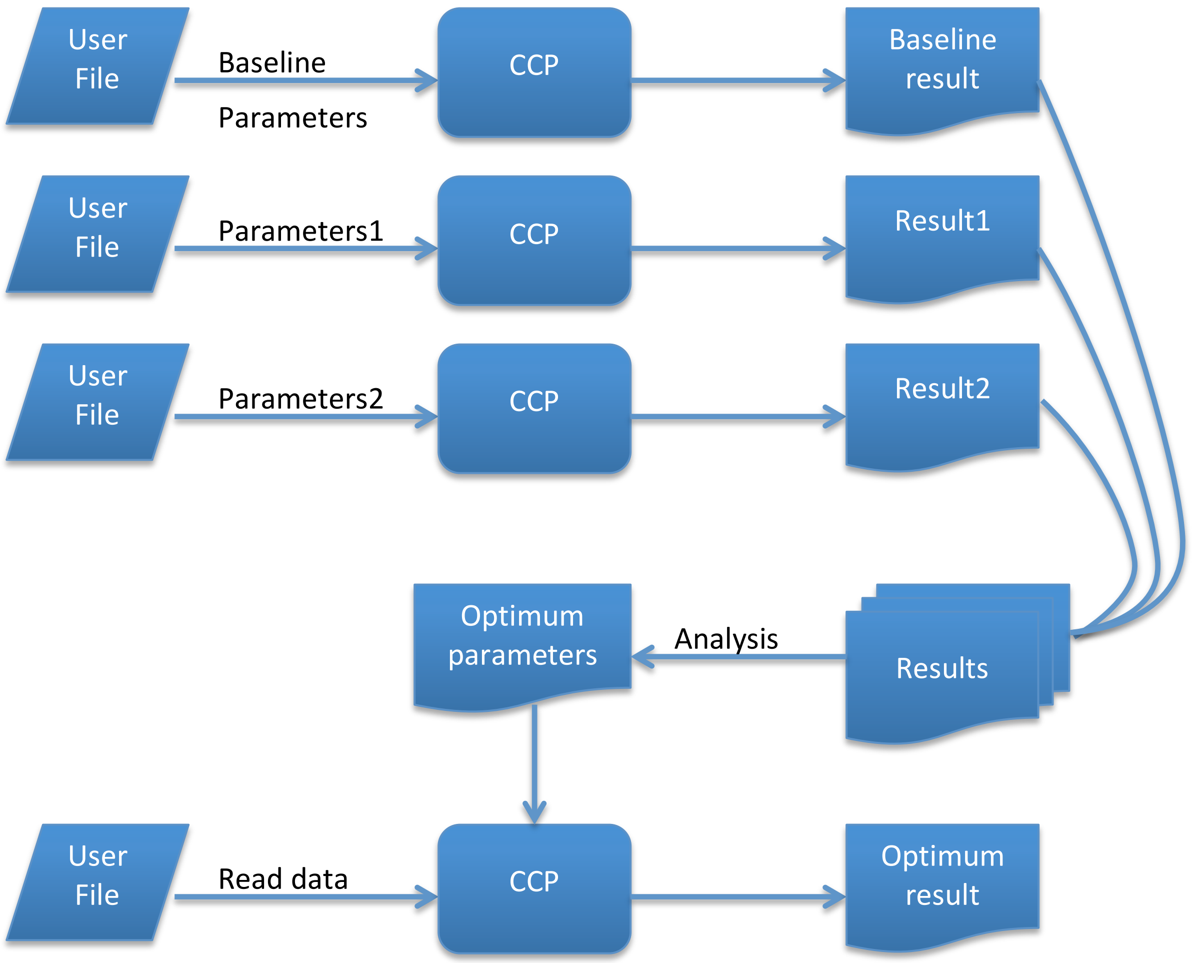

**Figure 1: Achieving better performance on reading a compressed data file**

## 2.2 Achieving better performance on writing compressed files

A user has an HDF5 file with compressed datasets. The user may use the CCP tool to recreate the data file using different parameters (different chunk sizes, different compression methods, …) to see the write performance for producing the data file and the size of the recreated data file. He can find the optimum parameters to use and apply them to tune his write application in the production of more data files. Figure 2 illustrates the overall process.

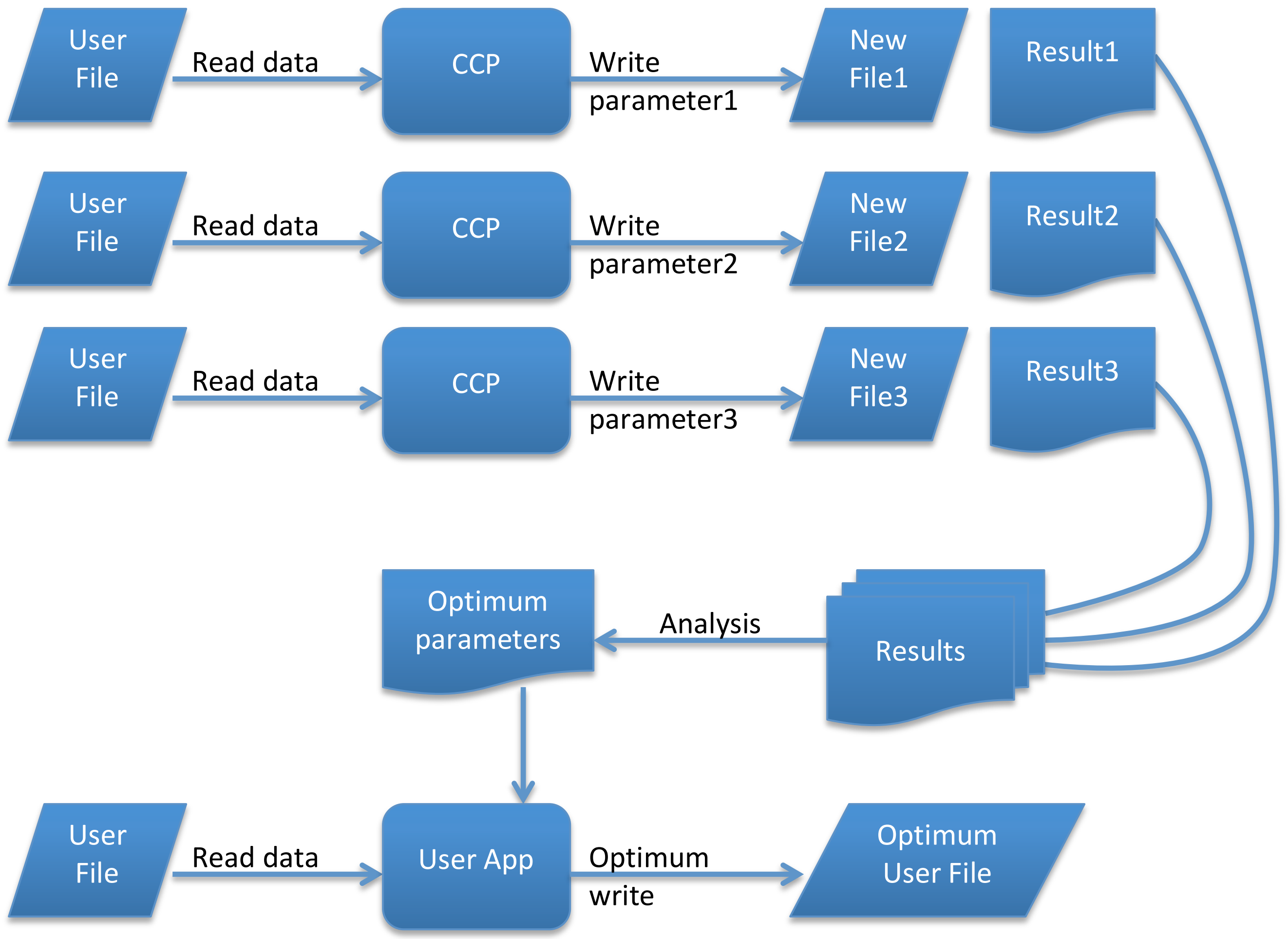

Figure 2: Achieving better performance on writing compressed files

### 2.3 General understanding of Compression and Chunking strategy

Users have been using the compression features of the HDF5 library such as the gzip and szip compressions in order to produce compressed data files that use smaller storage. It also helps file transfer if the transportation layer is slow. Users sometimes encounter poor performance during the write and read processes. They are often at a loss what the causes of poor performance nor how to improve the read and write speeds.

Therefore a tool that measures the read or write speed when compression is used with various parameters will help users to have a better understanding how the compression works so that users may efficiently store and retrieve data from HDF5 data files.

Since compression requires the use of chunk storage and the chunking scheme may affect the read/write performance substantially, the tool allows users to specify different chunking parameters during the measurement.

## 3 Requirement Specifications

This section describes the functionality of the Chunking and Compression Performance (CCP) tool to measure the performance of different HDF5 compression schemes with different parameters.

### 3.1 Function Definition

The CCP tool should accept user provided chunking and compression parameters to define the dataset storage and compression scheme; and do write and read operations on the datasets according to access orders and transfer buffer sizes specified by the user. It should report wall clock time taken to do the write or read operations on the datasets. The report of CPU execution time may be added in the future.

The compression parameters consist of different kinds of compression filters. The chunking parameters consist of chunk storage sizes and chunk cache sizes. The access patterns consist of order of dimensions accessed. The datasets parameters consist of the following,

- Data types
- fixed or unlimited dimensions
- fill values, fill time and allocation time
- cache and chunk cache sizes

All these parameters are specified in the following sections.

### 3.2 Filters of compression

The HDF5 library implements and supports data compression as filters. The tools should support all HDF5 pre-defined filters plus provision for user-defined filters. The following are current predefined compression filters supported by the library:

| | |
|---|---|
| `H5Z_FILTER_DEFLATE` | The gzip compression, or deflation, filter |
| `H5Z_FILTER_SZIP` | The SZIP compression filter |

| | |
|---|---|
| `H5Z_FILTER_NBIT` | The N-bit compression filter |
| `H5Z_FILTER_SCALEOFFSET` | The scale-offset compression filter |
| `H5Z_FILTER_SHUFFLE` | The shuffle algorithm filter |

## 3.3 Classes of data types

The tool defaults to use the BYTE data type initially. Integer and floating-point classes are then added. More HDF5 data types such as compound type may be added later.

## 3.4 Access Patterns

The tool should support access pattern by all legal combinations of orders of dimensions. For example, if the data file consists of 3-D (x,y,z) datasets, it should support all 6 access patterns of (x,y,z), (x,z,y), (y,x,z), (y,z,x), (z,x,y) and (z,y,x). The access pattern of (z,y,x) means the x-dimension varies the fastest while the z-dimension the slowest.

## 3.5 Memory buffer sizes

The memory buffer is used by the user to perform the read or write operation. The CCP tool should support any legal buffer sizes.

## 3.6 Dataset creation properties

The CCP tool should support user provided fill values, fill time and allocation times, chunk storage sizes during dataset creation and writing.

## 3.7 Dataset access properties

The tool should support user provided cache and chunk caches sizes during write and read operations.

## 3.8 User provided data file

### 3.8.1 Read performance only

The tool should, instead of creating its own data files for performance measurement, accept user provided data files to measure read performance.

### 3.8.2 Read and Write performance

The tool should support reading from user provided input data file to create user specified output data file and measure the read and write performance.

## 3.9 Performance statistics output

The tool should provide a summary of all parameters used for the performance measure and should provide 4 kinds of performance result:

- Statistics of writing data
- Statistic of file open, writing data, and file close

- Statistic of reading data
- Statistic of file open, reading data, and file close

Revision History

| | |
|---|---|
| *August 25, 2014:* | Rev 1: Function requirements drafted. |
| *November 13, 2014:* | Rev 2: Use cases added and implementation stages proposed. |
| *November 23, 2014:* | Rev 3: Motivation added and more features proposed. |
| *February 15, 2015:* | Rev 4: Abstract added. |