2\_User Guide\_A

Audience:

An HDF5 library application developer who has knowledge of HDF5 library APIs.

As a user creates an HDF5 file and manages its associated HDF5 objects, the HDF5 library allocates space to store the file’s metadata and raw data from the following entities in the library:

* Free-space manager

The HDF5 library’s free-space manager tracks free-space sections of different sizes when the library releases file space for metadata and user data. The free-space manager searches the free-space pool for a section of the appropriate size to fulfill the space request.

* Aggregator

The HDF5 library’s aggregator allocates a contiguous block of bytes designated specifically for the allocation of file metadata and another block for user data. The aggregator sub-allocates space from the appropriate block upon request.

* Virtual File Driver

The virtual file driver dispatches the individual driver’s allocation routine to get space upon request. The allocation is normally from the end of file.

The HDF5 library provides four file-space-handling strategies that a user can use to create a file. The strategies evolve from how the library employs the above three entities to fulfill file space requests:

1. H5F\_FILE\_SPACE\_ALL\_PERSIST

The HDF5 library tries to request file space first from the free-space manager. If it is not successful, the library tries allocation from the aggregator. If it is not successful, the library will allocate file space via the virtual file driver. For this strategy, the library’s free-space manager tracks the file space for metadata and user data that are released and the file space persists when the file is closed. This means the library saves the information to track the file’s free space at file closing, and the library can reuse the free space that exists in the file when it is re-opened.

1. H5F\_FILE\_SPACE\_ALL

This strategy is the library’s default file-space-handling strategy. It is the same as the previous strategy except that the space tracked by the free-space manager in the HDF5 file does not persist at file closing. This means the library does not save the information to track the file’s free space at file closing, and the library is unable to reuse the free space that exists in the file when it is re-opened. The space is lost as a result.

1. H5F\_FILE\_SPACE\_AGGR\_VFD

The HDF5 library tries to allocate file space first from the aggregator. If it is not successful, the library will request file space from the virtual file driver. For this strategy, the library does not the track the released file space for metadata and user data. That means the space is lost and cannot be reused.

1. H5F\_FILE\_SPACE\_VFD

The HDF5 library allocates file space only from the virtual file driver. The library does not the track the released file space for metadata and user data. This means the space is lost and cannot be reused.

The HDF5 library provides the following file creation property public routines that allow users to select the file-space-handling strategy for the file to be created. (See entry in HDF5 Reference Manual):

*herr\_t H5Pset\_file\_space(hid\_t fcpl\_id, H5F\_file\_space\_t strategy, hsize\_t threshold)*

There are three parameters to this public routine. The first parameter *fcpl\_id* is the file creation property identifier to be used in creating the file. The second parameter *strategy* is one of the four strategies described above. The third parameter *threshold* is the free-space section threshold used by the library’s free-space manager. This parameter is mainly for tuning purpose to improve library performance. The last section on *Performance Report for File Space Management* will address this further. A value of zero for *strategy/threshold* will retain the existing value in use.

The library provides another public routine that allows users to find out file space handling information for the file. (See entry in HDF5 Reference Manual):

*herr\_t H5Pget\_file\_space*(*hid\_t* *fcpl\_id*, *H5F\_file\_space\_t \* strategy*, hsize\_t \**threshold*)

The first parameter *fcpl\_id* is the file creation property identifier associated with the file. If the second parameter *strategy* is provided, the library will retrieve the existing file-space-handling strategy in use for the file and stores it in *strategy*. If the third parameter *threshold* is provided, the library will retrieve the existing free-space section threshold used by the library’s free-space manager and stores it in *threshold*.

The following sample coding shows the usage of these two public routines to create an empty HDF5 file *persist.h5* with file-space-handling strategy #1:

/\* Create file creation property template \*/

fcpl = H5Pcreate(H5P\_FILE\_CREATE);

/\* Set file-space-handling strategy #1 to use for the file \*/

/\* Retain existing free-space section threshold in use for the file \*/

H5Pset\_file\_space(fcpl, H5P\_FILE\_SPACE\_ALL\_PERSIST, (hsize\_t)0);

/\* Create the file with the file creation property *fcpl* \*/

fid = H5Fcreate(“persist.h5”, H5F\_ACC\_TRUNC, fcpl, H5P\_DEFAULT);

/\* The strategy retrieved will be #1 H5F\_FILE\_SPACE\_ALL\_PERSIST \*/

/\* The threshold retrieved will be 1 which is the library default \*/

H5Pget\_file\_space(fcpl, &strategy, &threshold);

/\* Close the file \*/

H5Fclose(fid);

The *h5dump* command line utility also allows the user to find out file space handling information in use for the file. See the following *h5dump* *–B* output for the file *persist.h5*:

HDF5 "persist.h5" {

SUPER\_BLOCK {

SUPERBLOCK\_VERSION 2

:

:

:

FILE\_SPACE\_STRATEGY H5F\_FILE\_SPACE\_ALL\_PERSIST

FREE\_SPACE\_THRESHOLD 1

}

:

:

The above output indicates that the file is using file-space-handling strategy #1 and a value of 1 for free-space section threshold, which is the library default.

2\_User Guide\_B

Audience:

An HDF5 library application developer who has knowledge of HDF5 library APIs,

and has some knowledge of the HDF5 library internals.

Each of the four file-space-handling strategies has its own benefits and drawbacks. The appropriate one to use depends on the user’s usage pattern of the HDF5 file.

Recall the two HDF5 files used in 1\_Primer, *not\_persist.h5* and *persist.h5*. By using H5P\_DEFAULT as the file creation property identifier when creating *not\_persist.h5*, the HDF5 librarywill automatically use file-space-handling strategy #2 for the file. User can generate the file *persist.h5* to use strategy #1 by following the coding sample in the previous section*.*

The discussion in 1\_Primer shows that strategy #1 has the benefit of reusing the tracked free space in the file while strategy #2 has the drawback of accumulating fragments of lost space in the file. The key factor contributing to the benefit of strategy #1 is due to the usage pattern of managing (adding/deleting) HDF5 objects across different settings, i.e. user creates/closes/reopens the file while managing HDF5 objects. Conversely, the fragmentation problem in using strategy #2 worsens as the user manages HDF5 objects from one create/close/reopen to the next.

Look at a different scenario (referred to as A) of creating *persist.h5* with strategy #1 but adds and deletes HDF5 objects all within one setting. For example, the user creates *persist.h5*, adds four datasets (*dset1, dset2, dset3* and *dset4*), deletes *dset2* and then adds *dset5* before closing the file. The output from *h5stat –S* shows the following:

Filename: ./persist.h5

Summary of storage information:

File metadata: 2409 bytes

Raw data: 4640 bytes

Amount/Percent of tracked free space: 117854 bytes/94.4%

Unaccounted space: 0 bytes

Total space: 124903 bytes

Use scenario A to create another file *not*\_*persist.h5* but with strategy #2. See the following *h5stat –S* output:

Filename: ./not\_persist.h5

Summary of storage information:

File metadata: 2216 bytes

Raw data: 4640 bytes

Amount/Percent of tracked free space: 0 bytes/0.0%

Unaccounted space: 117976 bytes

Total space: 124832 bytes

Note that the total space for *not\_persist.h5* is a bit smaller than *persist.h5*. For both files, the library’s free-space manager tracks the released free-space from the deletion of *dset2,* and reuses the free space for the addition of *dset5*. By comparing the size of file metadata for the two files, the greater amount of file metadata in *persist.h5* is due to the extra metadata needed to keep free space information persistent when the file closes. Thus, using strategy #2 for *not\_persist.h5* has some saving in file space than strategy #1 when managing HDF5 objects all within one setting.

Look at the last two file-space-handling strategies, #3 and #4. Both strategies do not use the library’s free-space manager in tracking released file space. Therefore, when managing (adding/deleting) HDF5 objects across different settings, a file using strategy #3 or #4 has the same drawback of accumulating fragments of lost space in the file like strategy #2.

This drawback also exists when managing HDF5 objects within one setting for a file using strategy #3 or #4. For example, create the HDF files *aggrvfd.h5* and *vfd.h5* with strategy #3 and #4 respectively, following scenario A. See the following *h5stat –S* outputs for the two files:

Filename: ./aggrvfd.h5

Summary of storage information:

File metadata: 2208 bytes

Raw data: 4640 bytes

Amount/Percent of tracked free space: 0 bytes/0.0%

Unaccounted space: 121936 bytes

Total space: 128784 bytes

Filename: ./vfd.h5

Summary of storage information:

File metadata: 2208 bytes

Raw data: 4640 bytes

Amount/Percent of tracked free space: 0 bytes/0.0%

Unaccounted space: 120272 bytes

Total space: 127120 bytes

Note that the total space for both files are bigger than that of *not\_persist.h5* (above figure ??) and *persist.h5* (above figure??). This is due to lost file space that cannot be reused.

However, strategies #3 and #4 have the benefit of saving file space when the usage pattern is adding HDF5 objects without deletion. For example, create the files *aggrvfd.h5* and *vfd.h5* with strategies #3 and #4 respectively, and then adds 4 datasets (*dset1, dset2, dset3, and dset4*) to each file. See the following *h5stat –S* output for the two files:

Filename: ./aggrvfd.h5

Summary of storage information:

File metadata: 2208 bytes

Raw data: 120640 bytes

Amount/Percent of tracked free space: 0 bytes/0.0%

Unaccounted space: 1936 bytes

Total space: 124784 bytes

Filename: ./vfd.h5

Summary of storage information:

File metadata: 2208 bytes

Raw data: 120640 bytes

Amount/Percent of tracked free space: 0 bytes/0.0%

Unaccounted space: 0 bytes

Total space: 122848 bytes

Compare the total space for the above two files with that of *persist.h5* and *not\_persist.h5* (see figure?? and figure ?? in 1\_Primer). All these four files are created with the same scenario but with different file-space-handling strategies. The total space for *aggrvfd.h5* and *vfd.h5* are smaller than that of *persist.h5* and *not\_persist.h5*. Note also that *vfd.h5* with strategy #4 has the greatest saving in file space.

Even though the file *aggrvfd.h5* with strategy #3 has less saving in file space than *vfd.h5*, it will have the benefit of better I/O performance due to the use of aggregator for file space request. The last section on *Performance Report for File Space Management* will give more information about I/O enhancement for this strategy.

Particularly aggravated when all within one setting all\_one\_setting () (~4k, 2k) or

add\_close\_adddelete() (this one is worser?hm…not really diff 2k but I have the bug in this one for VFD) : for #3, then #4

Bug in add\_close\_adddelete() for VFD