2\_User Guide\_A

Audience:

An HDF5 application developer who has knowledge of the HDF5 library API.

The HDF5 library performs file space management activities related to tracking free space and allocating space to store file metadata and data values in HDF5 dataset objects (raw data). Every HDF5 file has a file space management strategy associated with it that determines how the HDF5 library carries out these activities for the file.

Space Allocation

The HDF5 library includes three different mechanisms for allocating space to store file metadata and raw data:

* Free-Space Managers

The HDF5 library’s free-space managers track sections in the HDF5 file that are not being used to store file metadata or raw data. These sections will be of various sizes. When the library needs to allocate space, the free-space managers search the tracked free space for a section of the appropriate size to fulfill the request. If a suitable section is found, the allocation can be made from the file’s existing free space.

* Aggregators

The HDF5 library has two aggregators. Each aggregator manages a block of contiguous bytes in the file that have not been allocated previously. One aggregator allocates space for file metadata from the block it manages; the other aggregator handles allocations for raw data. The maximum number of bytes in each aggregator’s block is tunable.

If the library’s allocation request exceeds the maximum number of bytes an aggregator’s block can contain, the aggregator cannot fulfill the request. After space has been allocated from an aggregator’s block, that space it is no longer managed by the aggregator. Unallocated bytes continue to be managed by the aggregator. When an aggregator cannot fulfill an allocation request from the remaining space in its block, it requests a new block of contiguous bytes and any unallocated blocks that remain in the existing block become free space.

* Virtual File Driver

The HDF5 library’s virtual file driver dispatches requests for additional space to the allocation routine of the file driver associated with the HDF5 file. For example, if the H5FD\_SEC2 file driver is being used, its allocation routine will increase the size of the single file on disk that stores the HDF5 file contents to accommodate the additional space that was requested.

File Space Management Strategies

The HDF5 library provides several file space management strategies that control how it tracks free space and uses the free-space managers, aggregators, and virtual file driver to allocate space for file metadata and raw data. The strategies are:

**Strategy 1: H5F\_FILE\_SPACE\_ALL\_PERSIST (*also called* ALL\_PERSIST)**

With this strategy, the HDF5 library’s free-space managers track the free space that results from the manipulation of HDF5 objects in an HDF5 file. The free space information is saved when the HDF5 file is closed, and reloaded when the file is re-opened. The free space information **persist**s across HDF5 file sessions, and the free space managers are aware of free space sections that became available in any file session.

With this strategy, when space is needed for file metadata or raw data, the HDF5 library first requests space from the free-space managers. If the request is not satisfied, the library requests space from the aggregators. If the request is still not satisfied, the library requests space from the virtual file driver. That is, the library will use **all** of the mechanisms for allocating space.

The H5F\_FILE\_SPACE\_ALL\_PERSIST strategy offers every possible opportunity for reusing free space. The HDF5 file will contain extra file metadata information about tracked free space. The HDF5 library will perform additional “accounting” operations to track free space, and to search the free space sectors when allocating space for file metadata and raw data.

**Strategy 2: H5F\_FILE\_SPACE\_ALL (*also called* ALL)**

This strategy is the HDF5 library’s default file space management strategy. Prior to HDF5 Release 1.9.x, it was the only file space management strategy directly supported by the library.

With this strategy, the HDF5 library’s free-space managers track the free space that results from the manipulation of HDF5 objects in an HDF5 file. The free space managers are aware of free space sections that became available in the current file session, but the free space information is not saved when the HDF5 file is closed. Free space that exists when the file is closed becomes unaccounted space in the HDF5 file. Unallocated space in the aggregators’ blocks may also become unaccounted space when the session ends.

As with strategy ALL\_PERSIST, the library will try **all** of the mechanisms for allocating space with this ALL strategy. When space is needed for file metadata or raw data, the HDF5 library first requests space from the free-space managers. If the request is not satisfied, the library requests space from the aggregators. If the request is still not satisfied, the library requests space from the virtual file driver.

The H5F\_FILE\_SPACE\_ALL strategy allows free space incurred in the current session to be reused in the current session. There is no extra file metadata information about tracked free space in the HDF5 file. However, the HDF5 file will contain unaccounted space that can never be reused if free space exists when the file is closed. The HDF5 library will perform some additional “accounting” operations to track free space, but the amount of free space tracked and searched will probably be less than with the ALL\_PERSIST strategy, so the number of operations should be less.

**Strategy 3: H5F\_FILE\_SPACE\_AGGR\_VFD (*also called* AGGR\_VFD)**

With this strategy, the HDF5 library does not track the free space that results from the manipulation of HDF5 objects in an HDF5 file. All free space immediately becomes unaccounted space. Unallocated bytes in the aggregators’ blocks when the file is closed may also become unaccounted space.

With this strategy, when space is needed for file metadata or raw data, the HDF5 library first requests space from the aggregators. If the request is not satisfied, the library requests space from the virtual file driver. That is, the library will try the **aggr**egator and **v**irtual **f**ile **d**river mechanisms for allocating space.

The H5F\_FILE\_SPACE\_AGGR\_VFD strategy never reuses free space. Because small allocation requests can be satisfied from the aggregators’ blocks of contiguous bytes, this strategy will deliver better access performance for some file usage patterns. It may be appropriate when access performance is the highest priority and there are many small writes. Because there are different aggregators for file metadata and raw data, this strategy tends to co-locate file metadata more than some other strategies that can reuse free space scattered throughout the file.

**Strategy 4: H5F\_FILE\_SPACE\_VFD (*also called* VFD)**

With this strategy, the HDF5 library does not track the free space that results from the manipulation of HDF5 objects in an HDF5 file. All free space immediately becomes unaccounted space.

With this strategy, when space is needed for file metadata or raw data, the HDF5 library requests space from the **v**irtual **f**ile **d**river.

The H5F\_FILE\_SPACE\_VFD strategy never reuses free space. Because allocation requests go directly to the virtual file driver, this strategy is best suited for HDF5 files whose primary file usage pattern consists of writing large amounts of raw data to extend dataset object(s).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Summary of File Space Management Strategies** | | | | | | | |
| Full Name  *used with H5Pset\_file\_space* | | Short Name  *used with HDF5 utilities* | Track Free Space | | Allocate Space Using | | |
| across multiple sessions | within single session | free-space managers | aggregators | virtual file driver |
| H5F\_FILE\_SPACE\_ALL\_PERSIST | | ALL\_PERSIST | Y | Y | Y | Y | Y |
| H5F\_FILE\_SPACE\_ALL | | ALL | N | Y | Y | Y | Y |
| H5F\_FILE\_SPACE\_AGGR\_VFD | | AGGR\_VFD | N | N | N | Y | Y |
| H5F\_FILE\_SPACE\_VFD | | SPACE\_VFD | N | N | N | N | Y |

Specifying a File Space Management Strategy

The strategy for a given HDF5 file is specified when the file is created, and cannot be changed.

The HDF5 library provides the following file creation property routine that allows users to specify the file space management strategy to use when an HDF5 file is created later. (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 that will be used when the HDF5 file is created. 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 managers. This parameter is mainly for performance tuning purposes, and is discussed in more detail elsewhere. Passing a value of zero for the *strategy* or *threshold* indicates the corresponding parameter’s value should not be modified as a result of the call.

The library provides another public routine that retrieves the file space management information for an HDF5 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 an HDF5 file. If the second parameter, *strategy,* is not NULL, the library will retrieve the existing file space management strategy in use for the file and store it in *strategy*. If the third parameter, *threshold,* is not NULL, the library will retrieve the existing free-space section threshold used by the library’s free-space manager and store it in *threshold*.

The following code sample shows how these two public routines are used to create an empty HDF5 file, *persist.h5,* with file space management strategy ALL\_PERSIST:

/\* Create a file creation property template \*/

fcpl = H5Pcreate(H5P\_FILE\_CREATE);

/\* Set the file space management strategy \*/

/\* Don’t update the free-space section threshold \*/

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

/\* Create an HDF5 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 reports the file space management information for an HDF5 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 output indicates the HDF5 library will use file space management strategy H5F\_FILE\_SPACE\_ALL\_PERSIST and a free-space section threshold 1 (the default) when performing free space management activities on the HDF5 file *persist.h5.*

2\_User Guide\_B

Audience:

An HDF5 application developer who has knowledge of the HDF5 library API,

and also some knowledge of the HDF5 library internals.

Each of the four file space management strategies has benefits and drawbacks. The appropriate strategy depends on the HDF5 file’s usage pattern.

Recall the two HDF5 files used in 1\_Primer, *no\_persist.h5* and *persist.h5*. By using the default file creation property identifier (H5P\_DEFAULT) when creating *no\_persist.h5*, the HDF5 librarywill automatically use file space management strategy H5F\_FILE\_SPACE\_ALL for the file. The code sample in the previous section demonstrates how to create the file *persist.h5* that will be managed using the H5F\_FILE\_SPACE\_ALL\_PERSIST strategy *.*

The prior sections have shown that strategy ALL\_PERSIST has the benefit of reusing the tracked free space in the file across multiple file sessions, while strategy ALL has the drawback of accumulating unaccounted space in the file over multiple sessions. The key factor contributing to the benefit of strategy ALL\_PERSIST is the usage pattern of manipulating (adding/deleting) HDF5 objects across multiple sessions. The fragmentation and unaccounted space with strategy ALL increases with the manipulation (addition and deletion) of HDF5 objects across sessions.

Scenario C

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 management 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 management 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