

The Metadata Cache in HDF5

Changes in the HDF5 metadata cache since 1.6.3

References

A detailed overview of the metadata cache changes in HDF5 will be included in the HDF5 1.8 Users Guide, and the new API calls will be documented in the HDF5 1.8 Reference Manual. Draft versions of both of these documents are available online.

For a draft version of the overview, follow: http://hdf.ncsa.uiuc.edu/HDF5/doc_dev_snapshot/H5_dev/UG/UG_frame17SpecialTopics.html and then follow the link to Metadata Caching.

A draft version of the Reference Manual is at http://hdf.ncsa.uiuc.edu/HDF5/doc_dev_snapshot/H5_dev/RM/RM_H5Front.html

Look there for the entries on the new H5F and H5P API calls mentioned in the overview.

After the 1.8 release, be sure to use the 1.8 User's Guide and Reference Manual instead of the above links.

Definitions

Metadata – extra information about your data

- Two kinds:
 - Structural metadata
 - Stores information about your data
 - Example: When you create a group, you really create:
 - Group header,
 - B-Tree (to index entries), and
 - Local heap (to store entry names)
 - User defined metadata (Created via the H5A calls)
- Usually small – less than 1 KB
- Accessed frequently
- Small disk accesses still expensive

Definitions (continued)

Cache:

- An area of storage devoted to the high speed retrieval of frequently used data.

Metadata Cache:

- In HDF5, a module that tries to keep frequently used metadata in core so as to avoid file I/O.
- Exists to enhance performance.
- Limited size – in general, can't hold all the metadata all the time.

Cache Hit:

- A metadata access request that is satisfied from cache.
- Saves a file access.

Cache Miss:

- A metadata access request that can't be satisfied from cache.
- Costs a file access (several milliseconds in the worst case).

Definitions (continued)

Dirty Metadata:

- Metadata that has been altered in cache but not written to file.

Eviction:

- The removal of a piece of metadata from the cache.

Eviction Policy:

- Procedure for selecting metadata to evict.

Principle of Locality:

- File access tends not to be random.
- Metadata just accessed is likely to be accessed again soon.
- This is the reason why caching is practical.

Working set:

- Subset of the metadata that is in frequent use at a given point in time.
- Size highly variable depending on file structure and access pattern.

Example (Don't do this at home kids!)

If we:	Working set size is:	# of Cache accesses is:
1) Create four datasets A, B, C, and D with 1,000,000 chunks each in the root group of an empty file.	< 1 MB	< 50 K
2) Sequentially initialize the chunks using a round robin (1 from A, 1 from B, 1 from C, 1 from D, and then repeat until done).	< 1 MB	~30 M
3) 1,000,000 random accesses across A, B, C, and D.	~120 MB	~4 M
4) 1,000,000 random accesses to A only.	~40 MB	~4 M

Challenges peculiar to metadata caching in HDF5

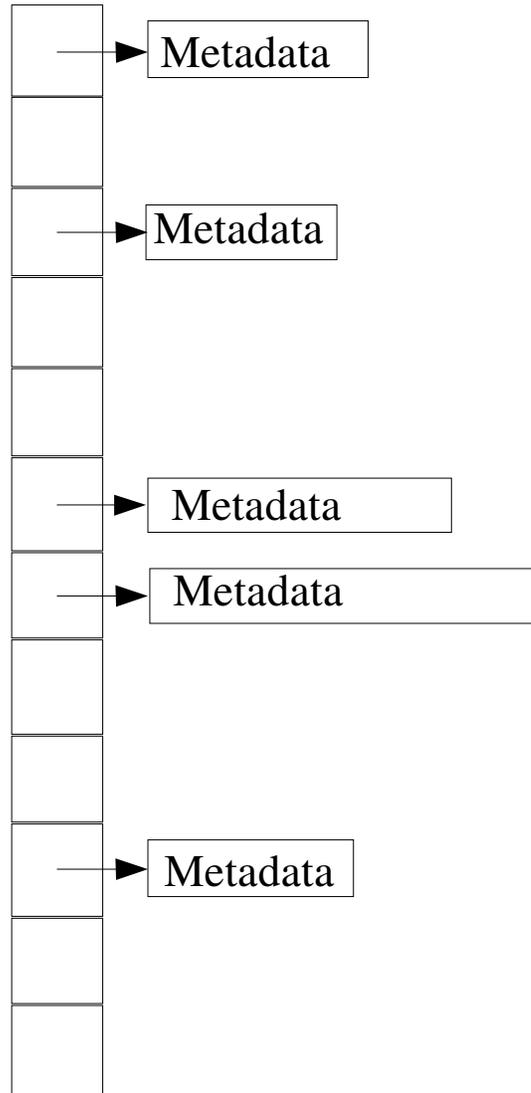
- 1) Wildly varying metadata entry sizes.
 - Most entries are less than a few hundred bytes.
 - Possible to create pieces of metadata of almost any size.
 - Entry sizes from bytes to megabytes exist in the wild.

- 2) Wildly varying working set sizes.
 - Less than 1 MB for most applications most of the time.
 - ~ 8MB for at least one in the wild.

- 3) Metadata cache competes with application programs for core.
 - Cache must be big enough to hold working set – otherwise hit rate and performance is poor.
 - Should never be significantly bigger lest it starve the user program for core.

The Metadata Cache in HDF5 1.6.3 and before

Hash
Table



No provision for collisions.

If a new entry hashes to the same location as an existing entry, the existing entry is evicted.

No other mechanism for eviction.

If the hash table is small:

- Poor performance as frequently accessed entries likely to hash to the same location, constantly evicting each other.
- Good cache size control, as entries don't sit in the cache long before being evicted.

The Metadata Cache in HDF5 1.6.3 and before (Continued)

If the hash table is big:

- Good performance, as frequently accessed entries unlikely to hash to the same location.
- Cache grows very big as entries are seldom evicted.
 - 100 times working set size observed in the wild.

Due to performance issues, must use large hash table.

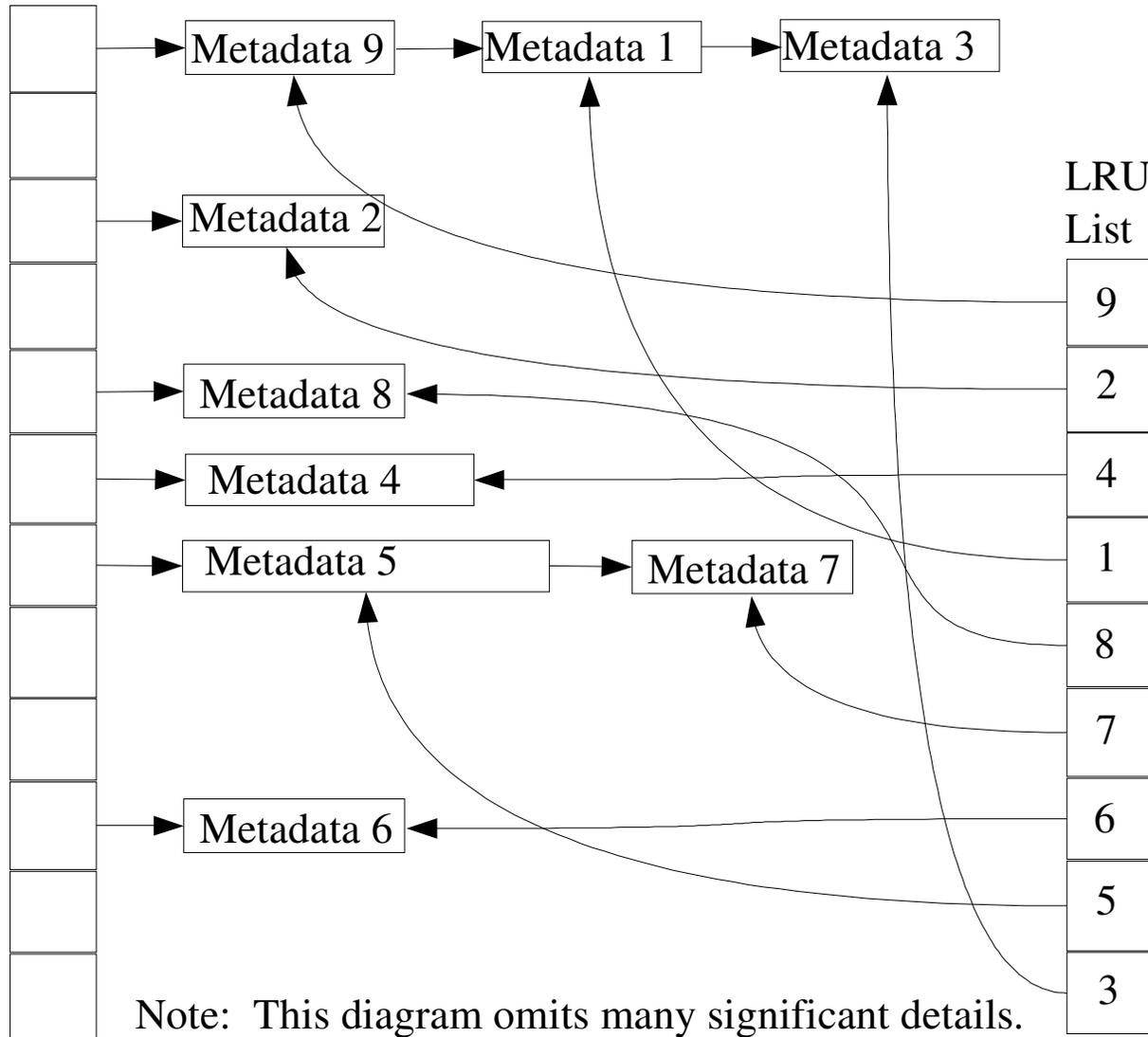
This version of the cache is:

- Fast.
- Economical in its use of computational resources.
- Adapts well to working set sizes seen to date.
- Has very inefficient use of core.

Inefficient use of core is unsustainable as HDF5 file size and complexity increases.

The Metadata Cache in HDF5 1.6.4 and 1.6.5

Hash
Table



Stores entries in hash table as before.

Collisions handled by chaining.

Maintains a LRU list to select candidates for eviction.

Maintains a running sum of the sizes of the entries.

Entries are evicted when a predefined limit on this sum is reached.

The Metadata Cache in HDF5 1.6.4 and 1.6.5 (continued)

Will grow to size limit, as no entries are evicted until the limit is reached.

Will not grow significantly beyond this limit.

Size limit is hard coded – 4 MB in 1.6.4, and 8 MB in 1.6.5.

- Library must be recompiled to increase these limits.
- Sizes were chosen to be large enough for all known applications.

Uses space much more efficiently.

- Good performance with 4 MB were the 1.6.3 cache used ~400 MB.

If working set sizes didn't vary wildly, this cache would be sufficient.

- But there are 1 – 8 MB working set sizes in the wild today.
- Larger variations are a matter of time.

Need adaptive cache resizing and API extensions for manual control of metadata cache size.

The Metadata Cache in HDF5 1.8

Metadata cache in version 1.8 is essentially identical to that in version 1.6.4 & 1.6.5, with two additions:

1) New API calls for:

- manual control of maximum cache size, and
- monitoring of actual cache size and current hit rate.

2) Code supporting adaptive cache resizing, with supporting API calls for control and tuning.

Adaptive cache resizing is enabled by default.

- Initial (and minimum) maximum cache size is 1 MB
- Should be sufficient for the vast majority of users.

Adaptive Metadata Cache Resizing in HDF5 1.8

Attempts to

- automatically detect the current working set size,
- set the maximum cache size equal to the working set size.

To the best of my knowledge, no-one has tackled this problem before.

- Some work with embedded processors is close.
 - Sections of processor cache dynamically enabled/disabled to obtain the desired performance / power consumption trade off.
- Let me know if you are aware of anything closer.

The problem breaks down into two sub-problems:

1) Size increase problem. Must:

- detect when the cache is too small, and
- select a size increment (some overshoot is OK).

2) Size decrement problem. Must:

- detect when the cache is too big, and
- select a size decrement (must not overshoot).

Adaptive Metadata Cache Resizing in HDF5 1.8 (continued)

Size increase problem is relatively easy – just monitor cache hit rate.

Every n cache accesses, check the hit rate.

If it is below a user defined threshold (i.e. 90%), increase the cache size by some user defined increment (i.e. a factor of two or a constant).

Repeat until hit rate is above the threshold.

Works well in most cases.

Doesn't work well when hit rate varies slowly with increasing cache size.

- Only seems to happen when working set size is very large.
- Probably not an issue for several years.

Adaptive Metadata Cache Resizing in HDF5 1.8 (continued)

Size decrement problem is harder. Best solution so far:

Track how long since each entry in the cache has been accessed.

Every n cache accesses check the hit rate.

If it is above some threshold, evict all entries that have not been accessed for more than some user specified number of cache accesses.

If this results in a cache size significantly below the maximum cache size, reduce the maximum cache size accordingly.

In my synthetic tests, the combination of these two algorithms works well.

How well they work in the field remains to be seen.

Adaptive Metadata Cache Resizing in HDF5 1.8 (continued)

Adaptive cache resize algorithms take time to react.

If the working set size varies too quickly, correct size is never reached.

This discussion has been greatly simplified.

See the references for detailed discussion. The User's Guide entry should be particularly useful.

HDF5 Metadata Cache Take Home Points

- 1) If your HDF5 files have relatively simple structure, you shouldn't notice the metadata cache changes.
- 2) If your HDF5 files are complex (i.e. huge groups, data sets with large numbers of chunks, or objects with large numbers of attributes), you:
 - Should see a considerable reduction in HDF5 library memory requirements between 1.6.3 and 1.6.4.
 - May see a performance drop between 1.6.5 and 1.8 caused by poor metadata cache hit rate while the adaptive cache resize code is adapting to your application. Solution – control cache size directly from your application.
- 3) If you work with complex HDF5 files and performance is an issue, please read the references. You will be happier.

Future Work

Future work on the metadata cache in HDF5 will be driven by where the bottlenecks are, and where the funding is coming from. However, the following items are on the to-do list:

- 1) Add code to collect metadata access sequences to facilitate tests of the adaptive cache resize algorithms against real world applications.
- 2) Algorithm development to repair some glitches that appear in extreme conditions with the current algorithms.
- 3) Determine if the adaptive cache resizing problem in HDF5 is really new. If it is, write a paper on it.

Acknowledgment

This presentation is based upon work supported in part by a Cooperative Agreement with the National Aeronautics and Space Administration (NASA) under NASA grant NNG05GC60A.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NASA.

Other support provided by NCSA and other sponsors and agencies.