h5repack: Improved Hyperslab selections for Large Chunked Datasets

Jonathan Kim

This document describes the cause of h5repack's low I/O performance issues with large chunked datasets and the update to improve performance. It compares performance measures with various test cases before and after the update.

1 Background

Prior to the update discussed in this paper (undertaken to resolve JIRA task HDFFV-7862), when any compression or layout option is used on the h5repack command line and a dataset is larger than 128MB in size, h5repack uses hyperslab to access a dataset instead of reading or writing the entire dataset at once.

h5repack's performance could be slow when certain chunking layouts were used. Sometimes reading a dataset was unnecessarily slow, sometimes writing a dataset. The worst case was when both reading and writing were unnecessarily slow.

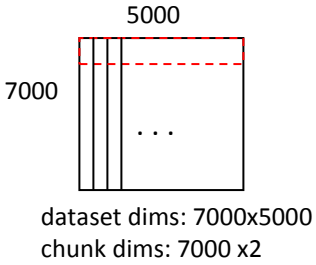
With this update, h5repack has been modified to take better advantage of chunking in reading and writing operations with hyperslab.

Related JIRA report

- HDFFV-7862 - Select data by chunk direction to improve performance in h5repack

2 Problem analysis

Slow performance occurred when the hyperslab selections used by h5repack involved small portions of multiple chunks in the dataset, as illustrated below.



The *red dotted box* represents the hyperslab selection, which is used for both read and write operations. The black dividers indicate the first three chunks in the dataset.

Even though only a portion from the each chunk is selected, the HDF5 library needs to read in the entire chunk. Therefore, in the above case, the entire dataset would be accessed repeatedly, once for each hyperslab selection as the hyperslab moves toward the bottom of the dataset.

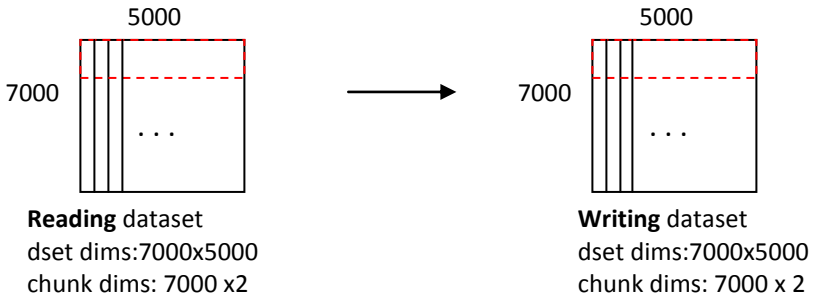
Prior to the update, performance suffered in situations like the above because the hyperslab was always calculated by data element as the base unit instead of considering the dataset's chunk layout.

2.1 Test cases, ranged from worst to best

This section describes the h5repack's performance in four cases before the improvement.

These cases were compiled to provide a baseline against which to measure h5repack's improved performance after the update. These tests also demonstrated that if the source and destination dataset chunk layouts are different, there is more improvement if h5repack's hyperslabs are aligned with the destination dataset chunk layout.

2.1.1 Case: Neither read nor write hyperslab aligned with dataset chunking



Reading dataset
dset dims:7000x5000
chunk dims: 7000 x2

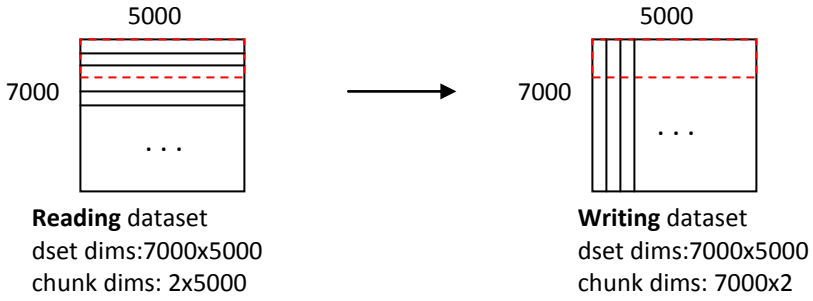
Writing dataset
dset dims:7000x5000
chunk dims: 7000 x 2

Test result: 2m 5sec

Reading h5repack's hyperslab is slow. Writing the hyperslab is slow.

This case yields the worst performance.

2.1.2 Case: Read hyperslab aligned with dataset chunking, write hyperslab not aligned



Reading dataset
dset dims:7000x5000
chunk dims: 2x5000

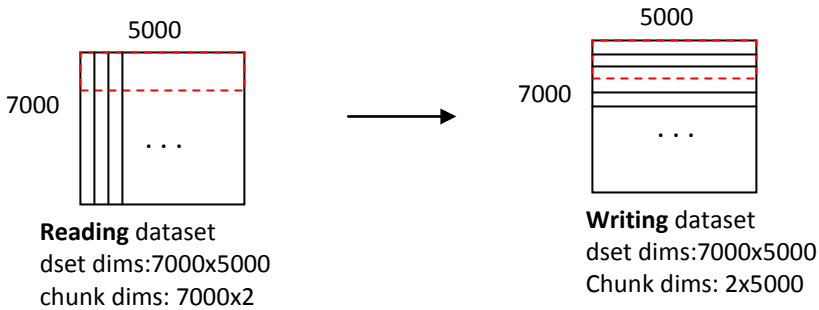
Writing dataset
dset dims:7000x5000
chunk dims: 7000x2

Test result: 1m 30sec

Reading h5repack's hyperslab is fast. Writing the hyperslab is slow.

This case yields better performance than the worst case, but can still be quite slow.

2.1.3 Case: Write hyperslab aligned with dataset chunking, read hyperslab not aligned



Reading dataset
dset dims:7000x5000
chunk dims: 7000x2

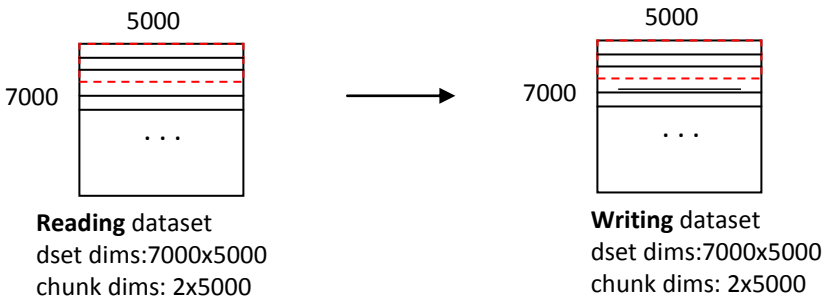
Writing dataset
dset dims:7000x5000
Chunk dims: 2x5000

Test result: 47 sec

Reading h5repack's hyperslab is slow. Writing the hyperslab is fast.

This case provides performance much closer to the best case, but can still be slow.

2.1.4 Case: Read and write hyperslabs both aligned with dataset chunking



Reading dataset
dset dims:7000x5000
chunk dims: 2x5000

Writing dataset
dset dims:7000x5000
chunk dims: 2x5000

Test result: 21sec

Reading h5repack's hyperslab is fast. Writing the hyperslab is fast.

This case provides the best performance.

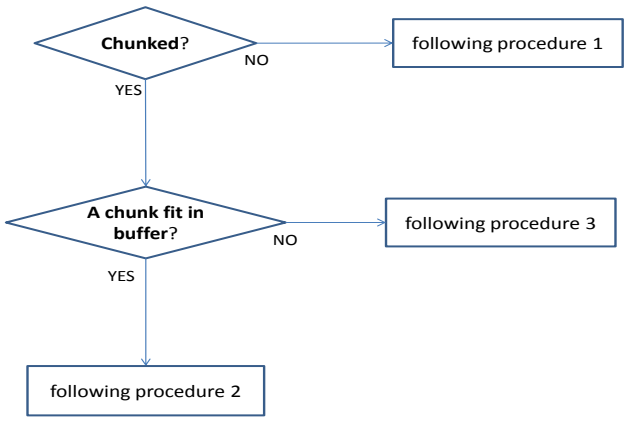
3 Update to improve performance with hyperslab

Once these performance issues were analyzed, it was determined that h5repack must be updated to make hyperslab selections more appropriately when working with large chunked datasets.

With this update, h5repack determines the dataset's chunk layout and aligns its hyperslab with the chunk layout instead of ignoring it.

Updated method to figure out a hyperslab :

- Calculates a hyperslab for big dataset in one of the following ways:



1. If the dataset is not chunked, build the largest rectangular hyperslab of **elements** that will fit into the buffer.

   The hyperslab calculation will start from the last dimension of the dataset. If the calculation hits the **boundary of the dataset's** dimension, the calculation continues processing with the next dimension of the dataset until the hyperslab buffer is full.

2. If the dataset is chunked and a chunk fits in the hyperslab buffer, build the largest rectangular hyperslab of **whole chunks** that will fit into the buffer.

   The hyperslab calculation will start from the last dimension (see h5dump dimensions output) of the dataset. If the calculation hits the **boundary of the dataset's** dimension, the calculation continues processing with the next dimension of the dataset until the hyperslab buffer is full.

3. If the dataset is chunked but a chunk does not fit in the hyperslab buffer, build the largest rectangular hyperslab of **elements** that will fit into the buffer.

   The hyperslab calculation will start from the last dimension of the chunk. If the calculation hits the **boundary of the chunk's** dimension, the calculation continues processing with the next dimension of the chunk until the buffer is full.

This update provides some improvement for all cases and the most improvement for the slowest cases described in Section 2.

3.1 Test results comparison before and after improvement

The four test cases from section 2 are reused here to compare performance before and after the update.

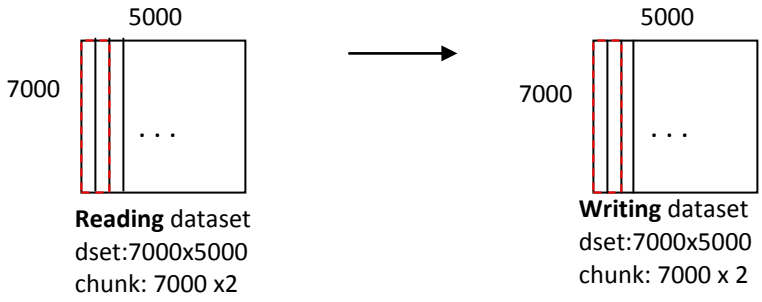
See the comparison table for the improvements. See the following subsections for case details.

h5repack performance before and after the update

	Before	*After*
Case1 (2.1.1 vs. 3.1.1)	2m 5sec	36 sec
Case2 (2.1.2 vs. 3.1.2)	1m 30sec	44 sec
Case3 (2.1.3 vs. 3.1.3)	47 sec	41 sec
Case4 (2.1.4 vs. 3.1.4)	21sec	20sec

3.1.1 Case 1

h5repack's hyperslab selection is aligned with both the reading and writing dataset chunk layouts.

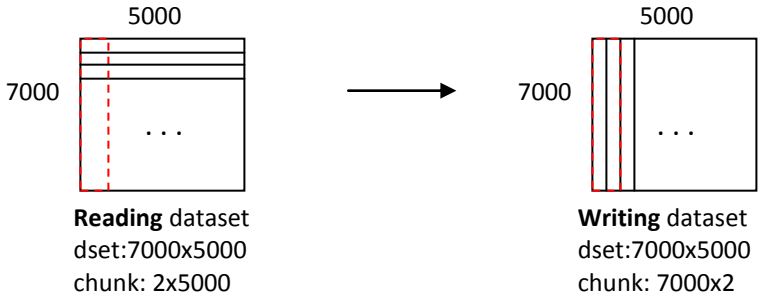


Test result: 36 sec

Reading h5repack's hyperslab is fast. Writing the hyperslab is fast.

This was the worst case before update.

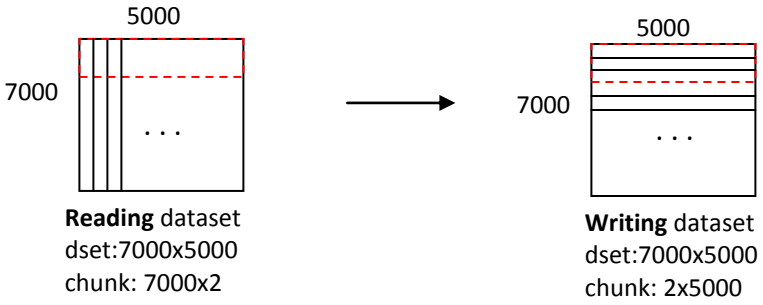
3.1.2 Case: Write hyperslab aligned with dataset chunking, read hyperslab not aligned



Test result: 44 sec

Reading h5repack's hyperslab is slow. Writing the hyperslab is fast.

3.1.3 Case: Write hyperslab aligned with dataset chunking, read hyperslab not aligned



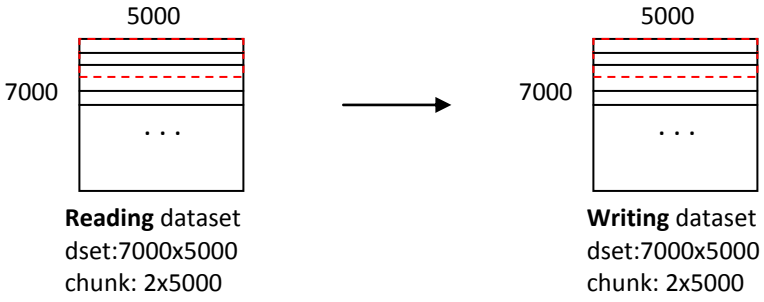
Reading dataset
dset:7000x5000
chunk: 7000x2

Writing dataset
dset:7000x5000
chunk: 2x5000

Test result: 41sec

Reading h5repack's hyperslab is slow. Writing the hyperslab is fast.

3.1.4 Case: Read and write hyperslabs both aligned with dataset chunking



Reading dataset
dset:7000x5000
chunk: 2x5000

Writing dataset
dset:7000x5000
chunk: 2x5000

Test results: 20sec

Reading h5repack's hyperslab is fast. Writing the hyperslab is fast.

4 Testing with user's data

This section presents two test cases with user data.

4.1 Test cases from the user reported in JIRA

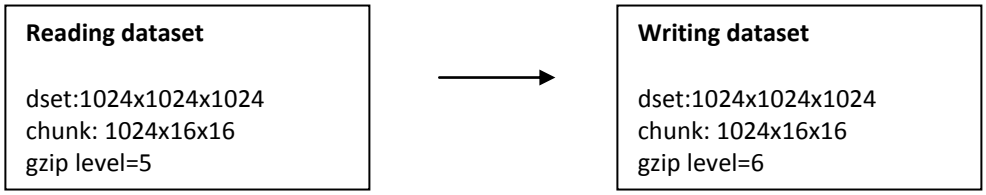
This test dataset is based on the user dataset in JIRA report HDFFV-7862. The original data size was too large, so a smaller dataset was created in a similar manner.

Performance comparison table before and after improvement

	Test1 (4.1.1)	Test2 (4.1.2)	Test3 (4.1.3)
Before	22 hours	92 m 4sec	30 m 26sec
After	15m 18 sec	13m 23 sec	13m 9 sec

4.1.1 Test 1

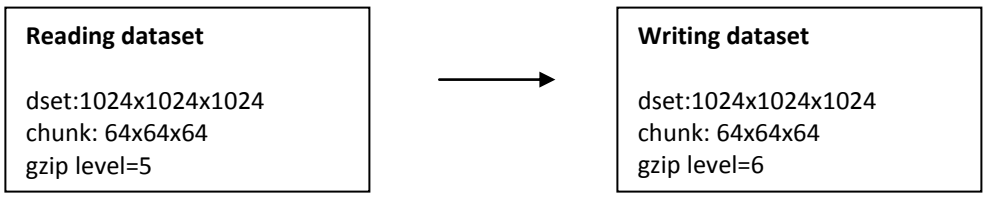
Command line: $ h5repack -f GZIP=6 srcfile1.h5 destfile1.h5



This use case is similar to case 1 in sections 2 and 3.

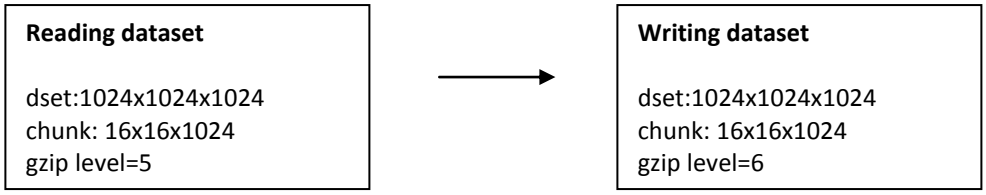
4.1.2 Test 2

Command line: $ h5repack -f GZIP=6 srcfile2.h5 destfile2.h5



4.1.3 Test 3

Command line: $ h5repack -f GZIP=6 srcfile3.h5 destfile3.h5



This use case is similar to case 4 in sections 2 and 3.

4.2 Test cases from a Help Desk user

This user was asking why changing the chunk layout took so long. The test file was provided by the user.

Performance comparison table before and after update

	Before	After
Test (4.2.1)	15 hours	38m

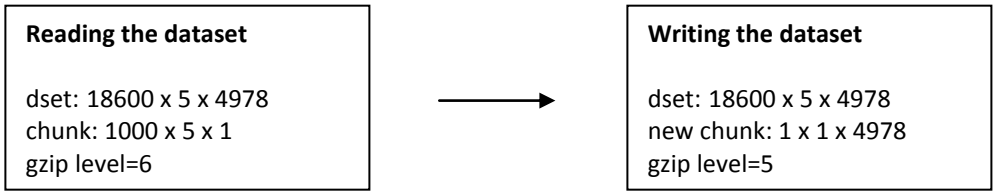
4.2.1 Test

Command line:

```
$ h5repack -f GZIP=5 -l <dataset>:CHUNK=1x1x4978 userfile.h5 userfile_repack.h5
```

The "userfile.h5" contains 24 datasets, 4 of which are large and chunked.

The chunk layout changing dataset is the biggest one.



This use case would be similar to case 2 from section 2.

5 Performance affecting factors considered and conclusions

This section highlights the factors that contributed to improved performance and provides brief conclusions.

- Improved hyperslab selection method
    - This was the major factor improving h5repack's performance in this update.
    - Calculated a hyperslab aligned with chunk layout.
    - If the source and destination dataset's chunk layouts are different, there is more improvement if h5repack's hyperslabs are aligned with the destination dataset's chunk layout.
- Increased hyperslab buffer size
    - The previous buffer size is 1MB and has been increased to 32MB to take advantage of modern hard disk caching capabilities.
    - This improved performance most when chunk size is smaller, so that more chunks exist in a dataset. This also improved performance when repacking non-chunked datasets.
- Increased threshold for declaring that a dataset is a large dataset
    - The previous threshold was 128MB and has been increased to 256MB. The hyperslab method is used for datasets larger than this threshold.
- About chunk cache size
    - Chunk cache size does not affect h5repack's performance much since data is not accessed repeatedly.

6 Future direction

- The updated method for getting a hyperslab will be extracted from the h5repack code as a separated common function for tools. This function can then be used to improve other tools. Consider h5diff, h5dump, and h5ls, for example, where the new function would improve the reading operation.

Revision History

February22, 2012: Version 1 draft 1 reviewed in tool team.

February27, 2012: Version 1 draft 2 updated to review in tool team.

February28, 2012:	Version 1 draft 3 updated to improve with doc team (Frank).
March 2, 2012:	Version 1 draft 4 updated to improve with doc team (Frank).
March 6, 2012:	Version1 to be preserved for future reference