H5edit Atomicity Performance Study

Albert Cheng

2013-10-15

http://www.HDFGroup.org

1. Introduction

This is a study of the performance times of the H5edit tool with different levels of the atomicity options with data files of different sizes.

The H5edit[1] tool is an HDF5 file editor. It supports commands to modify the contents of an existing HDF5 file. It enables HDF5 users to modify an HDF5 file without resorting to technical programming. Its intent is for small-scale modification of the file. Current version can modify the attributes of HDF5 objects such as datasets and groups.

1.1. The need of atomicity

It is important to users' production data files that the H5edit will execute the commands in an atomic manner, that is, it is either all success or no changes if there is any error. Otherwise, the HDF5 data file can be partially changed, which is not necessary desirable for all cases. Worse yet, if the H5edit fails in the middle of a command, the HDF5 file may be left in an unstable state, resulting in a total loss of access to the remaining information in the file. This is not an acceptable behavior for production files.

The H5edit tool creates and maintains a backup copy of the original data file being edited by the tool. The Atomicity option (--atomic) controls the manner the backup copy is managed. In case of user commands errors or system failures, the data file can be recovered from the backup copy by replacing the data file with the backup copy.

When the tool starts, after it has opened the data file successfully, it will make a backup copy of the data file before applying the input commands. If the tool encounters any error, the user may recover the data file from the backup copy.

1.2. Levels of atomicity

The H5edit tool provides three levels of atomicity: *no*, *yes*, and *inc*. The *no* level (--atomic==no) means no backup file is provided at all. This means the user does not care if the data file is partly changed or in an unstable state. He may already have a backup copy of the data file or he does not mind losing the data in the file.

The *yes* level (--atomic==yes) is the default setting for the H5edit tool and it means the tool makes a backup copy of the data file when it first opens it for editing. If the tool completes the editing session without any failure, it will remove the back up file after closing the data file. If the tool fails or is aborted, though the data file could be in an unstable state, the user may use the backup file to recover the data file.

The third level, *inc,* (--atomicity==inc) provides a finer level of backup support. One may see that the *yes* level of atomicity is an all-or-nothing backup support. A user may have completed many H5edit commands but has a minor typo mistake. It is annoying that he would have to redo all the editing commands. The *inc* (incremental) level instructs the H5edit tool to back up the data file after every successful edit command. This allows the above user to fix only the failed command and continues to complete the editing session.

[1] H5edit User Guide, v1.2.0

1.3. Performance concerns

Since the *yes* and *inc* levels of atomicity involve the creation and update of the backup file, it incurs extra I/O operations. File I/O operations are expensive comparing to computing. This study is to measure how much performance impacts among the three levels of atomicity are.

2. Performance Tests Setup

2.1. Time Measurement

The performance test is conducting by running h5edit on data file with different settings of atomicity to measure the impact of maintaining the backup file. The Unix Shell *time* command is used to measure the execution time of the h5edit session. Though the *time* command provides only 2 decimal points of execution time in seconds, it is sufficient for the purpose of this performance measurement as the differences of performance are expected to be substantial among the three atomicity levels.

2.2. Test data file of different sizes

Nine data files of different sizes are selected from files we have collected from NASA data center. They are chosen because each file has a size approximately double of the previous one. Below is a list of their file sizes:

File name	File size (MB)	Original NASA name
015MB.h5	15	SVDNB_npp_d20130727_t0000538_e0002179_b09046_c20130727063722389520_noaa_ops.h5
038MB.h5	38	VSCMO_npp_d20130601_t1200352_e1201594_b08259_c20130601182658726580_noaa_ops.h5
076MB.h5	76	IVIIC_npp_d20130601_t1200352_e1201594_b08259_c20130601135659498993_noaa_ops.h5
123MB.h5	123	GDNBO_npp_d20130727_t0000538_e0002179_b09046_c20130727061952493776_noaa_ops.h5
310MB.h5	310	GITCO_npp_d20130601_t1258533_e1300175_b08259_c20130601145518919477_noaa_ops.h5
549MB.h5	549	GDNBO-SVDNB_npp_d20130727_t0052066_e0057470_b09047_c20130801161207019107_XXXX_XXX.h5
2200MB.h5	2200	GDNBO-SVDNB_npp_d20130727_t0035017_e0057470_b09046_c20130731205748151135_XXXX_XXX.h5

2.3. H5edit Commands applied

Each data file is copied from the common repository to the local working directory. This ensures the same version of the data files is used. This is repeated with all three atomicity levels. The following is the Shell script used to run the performance tests and to collect the results posted by the *time* command.

```
for dfile in 015MB.h5 038MB.h5 123MB.h5 2200MB.h5 310MB.h5 549MB.h5; do
   for x in yes no inc; do
      cp data/$dfile $dfile
      rm -f "."$dfile".bck"
```

```
      time ./h5edit --atomic $x --command-file t_comm_file1 $dfile
   done
   rm -f $dfile
done
```

The command file t_comm_file1 contains the following nine H5edit commands. The reason that CREATE commands are used, is to make the HDF5 library more likely to read through all existing metadata, thus going through the whole file.

```
CREATE /newattrI32BE {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
CREATE /newattrF64LE {H5T_IEEE_F64LE (2, 2, 2) DATA {0.1E1, 0.2E1, -1.1e2, -
1.2e2, 2.1E-3, 2.2E-3, -3.1E-3, -3.2E-2}};
CREATE /newattr1 {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
CREATE /newattr2 {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
CREATE /newattr3 {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
CREATE /newattr4 {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
CREATE /newattr5 {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
CREATE /newattr6 {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
CREATE /newattr7 {H5T_STD_I32BE (1, 2, 3) DATA {11, 12, 21, 22, 31,32}};
```

2.4. Platforms measured

The test is conduced in three different platforms, namely Linux 32bit, Linux 64bit and Mac OS X systems. These three platforms are chosen because the NASA users commonly use them.

Platform type (hostname)	Machine details
Linux, CentOS 5.9, i686 (Jam)	Memory 16GB
Linux, CentOS 5.9, x86_64 (Koala)	Memory 12GB
Mac OS X 10.7.5 (Duck)	Memory 8GB

3. Results and Analysis

3.1. Linux system (jam)

File Size (MB)	Atomic=no (Sec)	Atomic=yes (Sec)	Atomic=inc (Sec)	yes/no	inc/no	inc/yes
15	0.004	0.302	0.214	76	54	0.7
38	0.003	0.102	0.539	34	180	5.3
123	0.004	0.338	4.676	85	1169	13.8
310	0.004	1.175	15.142	294	3786	12.9
549	0.005	2.243	30.339	449	6068	13.5
2200	0.004	56.329	363.612	14082	90903	6.5

3.1.1. Analysis of Linux system

Performance vs. file sizes

The overall execution time with atomic=no is constant. This is very much the normal execution time of the H5edit tool as there is no copying of the backup file. But it seems odd the execution time is independent of the size of the data file, as one would expect it should take more time to go through a bigger file. This can be explained by the design of Linux OS. Linux uses all available core memory to hold all file data and delays real disk I/O until all core memory is nearly used. Recall that the test setup makes a copy of the original data file to the local directory before running H5edit on it. This means, if the data file is not too big, it is in the core memory entirely when H5edit works on it. This explains the execution time is seemingly independent of the file size.

The overall execution time with atomic=yes is proportional to the sizes of the data file. This is expected. But the increase rate is slower when the size of the data file is from 15 to 549 MB; the file size ratio has increased 36.6 times (549MB/15MB) but the execution time has increased only 7.4 times (2.243s/0.302s). Again, this is due to the memory management of the Linux operating system. The Jam system has 16GB of core memory, file output time under 1GB is merely copying time between user and system memory. Between 549MB to 2200MB data files, there are only 4 times increase in file size but the time takes a 27 times jump. That is an indication that system memory is nearly exhausted and real disk I/O is happening. 2200MB/56s = 39MB/s is a more realistic disk I/O speed of the system. (Note that Jam has 16 GB of memory but it is also a heavily used machine.)

The overall execution time with atomic=inc is also proportional to the sizes of the data file but increases in a much faster pace. This is expected since there are 9 commands which means there are 9 copying of the data file to the backup file. Also, when file size increases from 15MB to 549MB (36.6 times), the process time increases from 0.214s to 30.339s (141 times).

In a nutshell, due to the testing procedure and Linux aggressive memory management to “hide” disk I/O speed, one should not make conclusion about performance time vs. file sizes, except to say in general, it take more time to process bigger data files.

Performance vs. atomicity levels
The last three columns of the table show the calculated ratios of different atomicity levels, namely *yes/no*, *inc/no* and *inc/yes*. They show substantial increase in execution time for both *yes/no* and *inc/no*. The increase ratio is also proportional to file size. This confirms the thinking that the backup file for restoration does cost execution time and the bigger the files are, the more execution time are needed.

On the other hand, the ratio of *inc/yes*, though large, stays very much constant for this machine. Again, this is due to Linux hiding disk I/O by keeping file data in core memory as much as possible. Therefore, even though when H5edit has to copy the data file to the backup file 9 times, it is merely data transfer between user and system memory. If the backup file can stay in memory, no disk I/O actually occurs during the execution of the H5edit tool. Not even once since at the end of the execution, H5edit removes the backup file because all editing commands pass. The Linux system would not waste time to push the content of a removed file to the disk.

3.1.2. Summary

It takes extra execution time to run the *yes* and *inc* atomicity levels to support the backup file for restoration and it takes even more time to support the *inc* level.

3.2. Linux64 system (Koala)

File Size (MB)	Atomic=no (Sec)	Atomic=yes (Sec)	Atomic=inc (Sec)	yes/no	inc/no	inc/yes
15	0.004	0.037	0.114	9	29	3.1
38	0.004	0.08	0.283	20	71	3.5
123	0.004	0.252	0.874	63	219	3.5
310	0.003	0.387	2.2	129	733	5.7
549	0.009	0.685	3.807	76	423	5.6
2200	0.008	22.44	34.531	2805	4316	1.5

3.2.1. Analysis of Linux64 system

This is also a Linux system though a more modern 64bit system. The performance results exhibit similar pattern as in the previous case of the Linux 32bit system

3.2.2. Summary

It takes extra execution time to run the *yes* and *inc* atomicity levels to support the backup file for restoration and it takes even more time to support the *inc* level.

3.3. Mac OS X 10.7 system (Duck)

File Size (MB)	Atomic=no (Sec)	Atomic=yes (Sec)	Atomic=inc (Sec)	yes/no	inc/no	inc/yes
15	0.219	0.512	0.764	2	3	1.5
38	0.286	0.958	1.912	3	7	2.0
123	0.306	6.27	5.455	20	18	0.9

310	0.22	7.009	14.496	32	66	2.1
549	0.3	11.946	61.056	40	204	5.1
2200	1.331	126.121	1207.505	95	907	9.6

3.3.1. Analysis of Mac OS X system

Performance vs. file sizes

The overall time with atomic=no is constant for file sizes up to 550MB. Though Mac system does not use the same aggressive memory management that Linux does, it still tries to hide disk I/O at a smaller scale as all operating systems do. As the file size increases from 549MB to 2200MB, about 4 times larger, the execution times of all three levels increase by 4, 10 and 20 times larger. This is not a surprise since the Mac platform has less memory than the other two Linux systems and is not able to keep both data and backup files in core.

Performance vs. atomicity levels

The last three columns of the table show the calculated ratios of different atomicity levels, namely *yes/no*, *inc/no* and *inc/yes*. They show increase in execution time for both *yes/no* and *inc/no*. The increase ratio is also proportional to file size. This confirms the thinking that the backup file for restoration does cost execution time and the bigger the files are, the more execution time are needed. Note that the increase ratios for this Mac platform (e.g., for the 2200MB data file are 95 and 907 times) are much smaller than the corresponding ones for the Linux platform (14082 and 90903 times). This is due to the Linux memory management masking the actual I/O speed of the atomic=*no* level. The ratios of the Mac platform are more accurate representation of the increase in execution times between different atomicity levels.

The ratio of *inc/yes* of this platform stays about the same for most file sizes and it is quite close to the expected ratio of 9 because there are 9 H5edit commands that cause 9 copying of the backup file.

3.3.2. Summary

It takes extra execution time to run the *yes* and *inc* atomicity levels to support the backup file for restoration and it takes even more time to support the *inc* level. The extra time to support the *inc* level over *yes* level is proportional to the number of H5edit commands.

4. Conclusion

It takes more execution times to support the backup file for restoration and the *inc* level takes even more time than the *yes* level. The impact depends on the file sizes and platforms. For smaller file size, the operating system's memory management can mask the disk I/O speed and lessen the increase in execution time. In Linux systems where it employs more aggressive memory management, it can mask the disk I/O of larger files if there is enough memory to keep the files in core.

The Backup VFD (Virtual File Driver)[2] has been proposed. It will optimize disk I/O operations to reduce the impact of the *inc* atomicity level. It will be good to implement it for the Mac OS X and the Linux platforms if the system's core memory is not likely to keep most NASA data files, together with the backup files, in core.

[2] Proposal of the Backup Virtual File Driver