Project Report

Evaluation of the deduplication file system "Opendedup"

SoSe 2012

Steffen Göttsch

Contents

Introduction

The subject of this project is evaluating the free deduplication file system "Opendedup". It is Open-Source Software, available at the website www.opendedup.org. It is based on Java and on FUSE, an system to provide a bridge to kernel interfaces.

A deduplication file system is a software which is set up in between the operating system and the hardware. A virtual drive is created, data stored on this will be managed by the deduplication file system. The deduplication process looks for redundant data, that is data segments which exist multiple times within the data. Only one copy of the duplicate data will be stored, for the other copies of this data only a pointer to the stored copy will be stored.

This way, the disk space needed to store the data may get reduced, depending if there is data that gets recognized as duplicated. Due to the operations needed to check for duplicated data, more CPU activity is needed for I/O operations, what might result in reduced read and write speed.

There are different methods of how data is deduplicated by various deduplication file systems. One difference is the time when the deduplication process happens. Post-process deduplication means that the data is stored on the disk as a whole first and the search for duplicated data happens afterwards. In-line deduplication means that the data gets checked for duplicate data before any data is actually stored on disk, which means that data recognized as duplicate data does never gets stored more than once. An obvious advantage of in-line processing is that the disk space used is never as big as with post-process and some useless writing operations are never done. A reason to use post-process nevertheless might be if one doesn't want to outsource the CPU-intensive check for duplicated data to a subsystem, where the data is stored. Opendedup has both options available, the standard method and the one I used during my tests is in-line deduplication.

Another important difference between deduplication file systems is the way duplicated data gets recognised. One very basic way is to only compare whole files, this is called single-instance-storage. Its usage is very limited by conceptuation. Other methods also check for duplicated data inside a single file and between files which are partly duplicate, but not as a whole. One method is fixed-block chunking, which cuts files from the beginning to the end into pieces of a given size and compares these with each other. Another, much more CPU-intensive, method is variable-block chunking, where the starting points and/or the size of the chunks is variable.

Subjects to evaluate

The two main aspects to be evaluated in this project are the storage space reduction one can gain by using opendedup and the extra I/O performance one needs in exchange. Beside this, what came to light in the evaluation process was a little insight into the mechanism Opendedup uses. Another subject on the side – which unfortunately came more into the center of attention during this project - was questions of stability of the file system and safety of the data.

As a overspanning, binding element lies the metaquestion if Opendedup is worth using. So first, is the compromise – storage space reduction versus reduced I/O performance – profitable, and second, are there other downsides in using it, like e.g. data safety issues.

Evaluation Process

Evaluation Environment

I tested the file system on two different systems. To get Opendedup running, FUSE and Java had to be installed first. Then I downloaded and installed the SDFS Binaries from the Opendedup homepage. Then I created a virtual volume wich was managed by Opendedup. On the creation of the volume, I also set the chunk size, that means how big the chunks are that are compared against each other to find duplicated data, using "--io-chunk-size <SIZE in kB>". On the second test series in an cluster in the University, I also set the storage place for the actual volume data and metadata wih the option "--base-path <PATH>" to make the data actually be stored on two different volumes for different test series. This volume was mounted in my home folder and the tests were run on this.

First I tested it in a virtual machine on my desktop PC at home, using Oracle VM VirtualBox 4.1.16. On this I installed Ubuntu 11.04 64 Bit. I assigned 2 GB RAM to this. The virtual volume had a size of 6GB and a chunksize of 4kB, the recommended chunk size for virtual machines. The Java heap size was remaining on the default size, wich is min 2GB and max 2GB.

The second tests were run on "sandy10", an I/O server at the scientific computing facilities of the university. The operating system was linux 64 bit. There were 16GB RAM available. The virtual volumes I created had 120GB size and a chunksize of 128kB. I made one volume on a HDD drive and one on a SSD drive. At my first tests I had issues with OutOfMemory Errors, so I changed the Java heap size in the /sbin/mount.sdfs to min 1,6GB and max 7GB by changing the -Xmx and Xms entry.

First simple Tests

First thing I am looking at is the storage space reduction. For this, it is neccesary to distinguish logical and physical size of data. Logical size is actually the size of the data, physical size is the disc space that is used to store this data.
For the latter, one detail is that the data stored is stored in segments of a certain size. That means if data doesn't fill a segment completely, the unused part of this segment is empty but cannot be used to store other data. So what we are looking at when talking about physical size is the disc space used including these empty, but not usable parts of the the disc space segments.

For seeing the logical file size, one can use the 'ls' command at the linux prompt. For seeing the physical file size as described above, one can use the 'du' command. One could also use 'ls -l' (ls -s?), this would give the disc space used without the free but unusable disc space. The proper thing to consider for this evaluation is the physical disc space as given by using the 'du' command.

The first test I did was storing mutliple copies of the same file on the file system. I used a music file for that, which normally doesn't have duplicate data inside itself, so I didn't expect a reduction in the size of this one file. The result was that one file had full file size like the original data, the other copies had zero file size. This lead me to the question where the overhead is stored in the process, an aspect I will probably take closer look at in the future. Overhead means the pointers to the data that is actually stored.

One thing I noticed during the first tests was that if I delete a file on the file system and afterwards put the same file on the file system again, the physical file size is zero. So, deleting the file on the file system does not delete the data chunk and hashtables, where the actual data and information about the data is stored. Of this I must be aware during the Evaluation process and clean up the chunk and hashtables before storing new data on the file system to get proper results.

Normally it is handled by the filesystem in the following way:
The chunk and hashtables remain the same until new data is stored on the file system. When new data is stored, the chunk gets checked if it contains data not used for a given period of time. Not used means that there are no files on the filesystem that point to these data segments. This data gets overwritten by the new data that is stored on the file system. One can set this period of time by using the following command:
setfattr -n user.cmd.cleanstore -v 5555:[time] [location of the file system]
But due to the way it works, this will not remove the data until new data is put on the file system.

So, to make sure the test results are right, there are two ways I am aware of:

- unmount the volume and delete all information about the volume, then mount a new volume before each test
- make sure the data I use is unique, that means similar data (in terms of how it is handled by the filesystem) is not stored on the volume before.

Storage space / deduplication mechanism tests with shellscript

The next tests I made considered the storage space reduction inside a single file whith chunks of repeated data. During these tests I gainded some insight on how the file system works. First I ran these Tests on the virtual machine at home. Later I repeated the Tests on sandy10, with identical results.

I wrote a shell script (named "OpenDedupEvaluation") that creates test files to illustrate some cases of deduplication. I will list each of these and explain the results. For all except one test I used chunks of data that fit the chunk size of the test volume I created, wich was 4k. I used ten chunks of data on each test case. I used a marker to make each chunk unique that was supposed to be unique, starting at 1000 and increased by 1 each time I wanted a new (unique) chunk. Each chunk contains this marker and (chunksize – 4) times "x". So, the only difference between the chunks is that marker. The file created is a plain text ASCII file, so each character has the size of one byte.

Test case 100% unique data:
Each chunk has its own unique marker (that means the marker gets increased by 1 after a chunk is written). The physical size of this file was 40k, which is 10 times the chunk size. No storage space reduction.

Test case 100% duplicated data:
The same marker is used for each chunk. The physical file size was 4k, that means the chunk is stored only once, the other 9 chunks are stored only by pointing to this one chunk.

Test case 50% duplicated data:
The first 5 chunks get each its own marker, the other 5 chunks use the same marker as the 5th chunk. The physical file size was 20k because each of the unique chunks is stored as a whole, the repeated chunks are only stored as pointers.

Test case 100% duplicated data, not fitting the chunk size of the test volume:
Created like the other 100% duplicated data test case, but adding one "x" for each chunk. So the chunk size in this case is 4k + 1. The physical file size was 40k, that means no storage reduction was done by the file system.

These examples illustrate very good how the file systems works. It obviously cuts a file from beginning to end into parts as big as the chunksize of the volume and compares these chunks. It only dedups if they are exactly the same. This is shown very clear with the first test case, where only one or two bytes differ, but no physical size reduction occurs. In the second case all chunks are exactly the same and it dedups well.
It does not notice data as duplicated data, if a duplicated piece of data starts/ends anywhere else than at the cutting points, as the last test case shows. If it would, then the last test case would have been stored with a much smaller physical size (close to 4k).

I/O performance tests with IOzone

The other big part of the project was to evaluate the I/O permormance on SDFS volumes using IOzone. Iozone is a benchmark tool, that does a lot of I/O operations automatically and records the speed of these operations.

The first tests were run on the volume in the virtual machine. The command line used was in the form of this:

```
iozone -a -n 10m -g 250m -q 10m -y 16k -+w 0 -f /home/s/vol4k/tmp -i 0 -i 1 -i 2 -i 9 -i 10 -R -b
/home/s/project/iozone_results/000p1.xls
```

The options used have these effects:

-a:	runs automatic mode, which does a number of things automatically.
-n:	minimum file size tested
-g:	maximum file size tested
-y:	minimum record size tested
-q:	maximum record size tested
-+w:	% dedupable
-f:	the name and location of the test file (in the location of the virtual volume to be tested)
-i:	the tests to be run (0=write/rewrite, 1=read/re-read, 2=random-read/write, 9=pwrite/Re-pwrite, 10=pread/Re-pread)
-R:	Generate excel report
-b:	name and location of the excel file

The Results are the excel files stored in the "VM" folder, using mostly the options stated above, with differences as following:
000p1: on the Opendedup volume, with -+w 0 (0% dedupable)
050p: on the Opendedup volume, with -+w 50 (50% dedupable)
100p: on the Opendedup volume, with -+w 100 (100% dedupable)
standard_000p: on the normal filesystem, with -+w 0 (0% dedupable)
standard_100p: on the normal filesystem, with -+w 100 (100% dedupable)

On the normal file system the dedupable % shouldnt make any difference, because it doesn't check for duplicate data, but I included it in the tests for completeness' sake. One thing I noticed when just looking at these tests on the normal filesystem was there were many irregularities when looking at the values. By that I mean when there seems to be a pattern of similar values in the surroundings and one value doesn't fit that, for instance in "standard_000p.ods" the field in Re-Writer Report with 10240kB filesize and 2048kB record size is exceptionally low compared to the surrounding values (only about one third of the others), where all the other values are relatively close together (only differ by about 20%).

That leads me to the assumption that the accuracy of these first tests is not really high. Not a really surprising occupation, because these tests where run on a virtual machine and in quite a small scale. I was looking forward to the tests on the university cluster.
When comparing the tests on the Opendedup volume to the ones on the normal file system, the normal file system seems to be much faster, in writing roughly about three times as fast and in reading about ten times at fast. The fact that the Opendedup filesystem seems to be slower was anticipated by me, but the fact that the differences are so huge are suspicious. Together with the noticed inaccuracies, I would not interprete too much into these numbers.

Next came the tests on the sandy10 at the university cluster.
I started with an ambitiously large set of test runs in mind, but due to the many problems that occured in the process, I had to reduce it greatly to get the project done within a reasonable time. On the first bigger tests I had running times of about two days for a single test, so the time to run a single test was really an issue, especially since many tests crashed and I had to redo them. These problems had such a great impact on the project, that I saved an own section in the report for them. But first, in this section, I will describe and interprete the final, reduced set of test runs.

I tested on a SSD and on a HDD volume. I used 1MB and 16 MB record size and file sizes from 4GB to 32 GB on each test run. On the Opendedup volume, I tested with 0%, 50% and 100% dedupable data (using the -+w option of IOzone). On the normal volume, I tested only 0% dedupable data. On the normal file system I also made a bigger test run on the SSD with file sizes from 250MB to 64GB. Additionally, I ran tests with a record size of 4kb on the Opendedup volume on the SSD, but only with file sizes from 100MB to 1,6GB, because the tests with this small record size ran very slow.

For these tests on the cluster, I connected to the cluster with the program "PuTTY". At first I started PuTTY multiple times to be able to mount a volume and run iozone at the same time. Later I switched to using only one instance of PuTTY and used the linux program "screen", which handles different console instances in one actual console window (The one I see in PuTTY). It also stores the console sessions when I log out, which comes very handy, so I can switch my PC off if I ran a test with very long running time and the test runs on at the cluster.

The command line looked something like this:
```
./iozone -a -n 4g -g 32g -r 1m -r 16m -+w 0 -f /home/goettsch/volSSD/tmp -i 0 -i 1 -i 2 -i 9 -i 10 -R
-b /home/goettsch/logs/SSD/000poSSD.xls
```

Most of the options are already explained above. The only new option is -r, which specifies the exact record size for the test, or multiple record sizes if used more then once. This is different from the -y and -q option from the first test, which specifies a range, in which various record sizes are tested. As I wanted to reduce the running time of the tests, I only used the three mentioned record sizes 4kB, 1MB and 16MB.

The tests with 4kB record size ran quite slow, about 80.000 kB/s for write and re-write, about 120.000 kB/s for read and re-read, and only about 38.000 for random write and random read.

For the tests with 1MB and 16MB record size:
On the normal file system, the write speed for big file sizes – 16GB and 32 GB - was about 180.000 kB/s on the SSD and 90.000 kB/s on the HDD. This seems to be the actual physical write speed on the hard drive. Read speed for these file sizes was about 260.000 kB/s on the SSD and 90.000 kB/s on the HDD. For smaller files, these I/O operations were much faster, this is best seen on the larger test on the SSD (filename 000pnSSD-f.ods).

We can see extremely fast operations for the small files here, that are actually much faster then the possible physical write/read speed. Then at about 4GB filesize for write and 16GB for read, the speed slows down to about 10% of the previous speed, which is about the maximum physical write/read speed. This can be explained by the assumption that for small files the data is not actually written on disk but only stored in the cache, which has much faster access time than the physical storage media.

At the tests on the Opendedup volumes, both on the HDD and the SSD, the tests with 0% dedupable were very fast, around 900 kb/s. My first Assumption was that all the data, that should be written on disk, is actually only stored in the cache. To verify this, I ran an extra test with the iozone options -e, which includes flush in timing calculations, and -o, which should force the filesystem to write the files to disk immediately using the O_SYNC flag. The results stayed the same, so first I was assuming that Opendedup Ignores the O_SYNC flag.

I ran another test using the -w option, which leaves the temporary test files present on the filesystem after the test, to be able to check the file size actually stored after the test. The result was that the physical file size of the data was zero, which means that similar data was on the filesystem before and gets completely deduplicated (only a pointer to the first copy of this data on the filesystem gets stored). I also checked the basepath of the volume, and the data stored there (the chunks and the metadata) was much smaller than the file size used in this test, so obviously there happened much deduplication even inside a single file. This explains that the speed is higher than the possible physical writie speed of the hard drive, because only a small portion of the data gets actually stored on disk. So the concluding assumption is that the IOzone options for the deduplication rates do not work properly.

Strangely, the tests with 50% and 100% dedupable data were different in the way that the Writer Report were slower than the test with 0%. For 4GB files it was about 500MB/s and towards the tests with bigger files it slowed down to about the speed one would expect for actually physically written data, at 32GB file size it was about 180MB/s. For the Re-writer Report and the Random Write Report, the speed was very high, about 1GB/s. The Reader Reports were mostly in a speed range from 100 to 150 MB/s.

Problems during the Tests

As previously mentioned, quite some problems occured during my tests, which significantly hindered the process of the Evaluation. The first occured often during my first tests on the virtual machine. When I ran some tests, unmounted the volume and tryed to remount it again or mount a different volume, the following Error came and I wasn't able to mount any volume:

```
fuse:fork OutOfMemory Error
```

After a restart of the operating system I could mount volumes again.

Once I wanted to stop the IOzone test run and killed the process. The Opendedup volume also crashed and I was not able to mount it again. This was a first strong hint that data stored on such a file system is maybe not safe.

At multiple occasions during the first tests on sandy10 with IOzone I had this error in the mount screen, after which the volume was unmounted:

```
Exception in thread "Thread-11"
Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread
"ActionDistributor-0"
Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread
"FileManager-0"
^CJava HotSpot(TM) 64-Bit Server VM warning: Exception java.lang.OutOfMemoryError
occurred dispatching signal SIGINT to handler- the VM may need to be forcibly terminated
```

This was the error in the IOzone screen at these crashes:

```
Error reading block 2459 7f9b00300000
read: Bad file descriptor
```

I tried to solve this problem with assining more memory to Opendedup. I raised the java stack size by changing the entries -Xmx and -Xms in the /sbin/mount.sdfs to -Xmx7000m and -Xms1600m.

After I changed the stack options in IOzone, the OutOfMemory error didnt happen any more. Instead, there were often this error in the mount screen:

```
/sbin/mount.sdfs: Zeile 4: 22196 GetÃ¶tet              /usr/share/sdfs/jre1.7.0/bin/java
-Djava.library.path=/usr/share/sdfs/bin/ -Dorg.apache.commons.logging.Log=fuse.logging.FuseLog
-Dfuse.logging.level=INFO -server -Xmx7000m -Xms1600m -XX:+UseG1GC -XX:
+UseCompressedOops -classpath /usr/share/sdfs/lib/truezip-samples-7.3.2-jar-with-
dependencies.jar:/usr/share/sdfs/lib/jacksum.jar:/usr/share/sdfs/lib/trove-
3.0.0a3.jar:/usr/share/sdfs/lib/slf4j-api-1.5.10.jar:/usr/share/sdfs/lib/slf4j-log4j12-
1.5.10.jar:/usr/share/sdfs/lib/quartz-1.8.3.jar:/usr/share/sdfs/lib/commons-collections-
3.2.1.jar:/usr/share/sdfs/lib/log4j-
1.2.15.jar:/usr/share/sdfs/lib/jdbm.jar:/usr/share/sdfs/lib/concurrentlinkedhashmap-lru-
1.2.jar:/usr/share/sdfs/lib/bcprov-jdk16-143.jar:~/java_api/sdfs-bin/lib/commons-codec-
1.3.jar:/usr/share/sdfs/lib/commons-httpclient-3.1.jar:/usr/share/sdfs/lib/commons-logging-
1.1.1.jar:/usr/share/sdfs/lib/commons-codec-1.3.jar:/usr/share/sdfs/lib/java-xmlbuilder-
1.jar:/usr/share/sdfs/lib/jets3t-0.8.1.jar:/usr/share/sdfs/lib/commons-cli-
1.2.jar:/usr/share/sdfs/lib/simple-
4.1.21.jar:/usr/share/sdfs/lib/jdokan.jar:/usr/share/sdfs/lib/commons-io-
1.4.jar:/usr/share/sdfs/lib/sdfs.jar fuse.SDFS.MountSDFS $*
```

When I tried to remount the volume after these crashes, I had this error:

```
[goettsch@sandy10 ~]$ sudo mount.sdfs -v vol4kHDD -m vol4kHDD
[sudo] password for goettsch:
Running SDFS Version 1.1.5
reading config file = /etc/sdfs/vol4kHDD-volume-cfg.xml
Loading ##################java.lang.ArrayIndexOutOfBoundsException: 2
      at org.opendedup.collections.CSByteArrayLongMap.getMap(CSByteArrayLongMap.java:86)
      at org.opendedup.collections.CSByteArrayLongMap.put(CSByteArrayLongMap.java:553)
      at org.opendedup.collections.CSByteArrayLongMap.setUp(CSByteArrayLongMap.java:297)
      at org.opendedup.collections.CSByteArrayLongMap.init(CSByteArrayLongMap.java:73)
      at org.opendedup.sdfs.filestore.HashStore.connectDB(HashStore.java:165)
      at org.opendedup.sdfs.filestore.HashStore.<init>(HashStore.java:78)
      at org.opendedup.sdfs.servers.HashChunkService.<clinit>(HashChunkService.java:61)
      at org.opendedup.sdfs.servers.SDFSService.start(SDFSService.java:59)
      at fuse.SDFS.MountSDFS.main(MountSDFS.java:148)
00:00:41.366     main  INFO [fuse.FuseMount]: Mounted filesystem
fuse: bad mount point `vol4kHDD': Der Socket ist nicht verbunden
00:00:41.375     main  INFO [fuse.FuseMount]: Filesystem is unmounted
```

After some crashes, I looked at the Usage of the RAM with the program "top".
There were 2 java processes running, wich occupied 31% and 26% of the memory. As noone else was working on sandy10 at that time, this had to come from my tests. It seems that Opendedup stays in the memory after it crashes, which is not a good thing.

I looked a bit more at the memory usage of Opendedup. When a volume is mounted, at first it (the java process, which I assume belongs to OpenDedup) uses very little memory. When files are written on it, it uses about 45% of the memory. This memory usage is not only during the writing process, but also stays if the system is idle for a while. When unmount the volume normally with "umount [mountpoint]", the java process gets away.

Finally, here I list some other errors that I had during the tests on sandy10:

Another error that occured was this one:

```
18:25:30.004 Thread-10 ERROR [fuse.SDFS.SDFSFileSystem]: java.lang.NullPointerException
java.lang.NullPointerException
      at org.apache.commons.collections.map.LRUMap.moveToMRU(LRUMap.java:194)
      at org.apache.commons.collections.map.LRUMap.get(LRUMap.java:178)
      at org.opendedup.sdfs.io.SparseDedupFile.getWriteBuffer(SparseDedupFile. java:414)
      at org.opendedup.sdfs.io.DedupFileChannel.writeFile(DedupFileChannel.jav a:246)
      at fuse.SDFS.SDFSFileSystem.write(SDFSFileSystem.java:602)
      at fuse.Filesystem3ToFuseFSAdapter.write(Filesystem3ToFuseFSAdapter.java :338)
```

On another occasion I had this error in IOzone, but only in IOzone, the volume was normally mounted afterwards:

```
Error reading block 1882188 7ff42cb00000
read: Permission denied
```

On another test run, IOzone didnt react any more, so I killed the process. "umount" afterwards didnt work, The error said "volume is busy", so first I killed the mount process. The volume was still busy afterwards, so I killed the java process too. Afterwards I wasnt able to remount the volume.

On some occasions had this error when I tried to mount a volume again on a previously used mountpoint after a crash:
Fuse: bad mount point 'volSSD': Der Socket ist nicht verbunden

Most of the errors occured with bigger test sets, that is with more file sizes and record sizes. So it seems to be that a big amount of data written on an Opendedup volume might cause errors or at least favor the propability.

Conclusion

For the subjects I wanted to evaluate, I sadly can't say any good conclusion to the I/O performance due to the strange data I got, which seems to indicate that IOzone doesn't give undeduplicatable data when it should, at least not for the deduplication mechanics Opendedup uses.

For the Storage space reduction, as I showed in my tests with the shell script, it works very good if the data is has a corrsponding pattern to the fixed-block deduplication mechanism, that is if it has chunks of duplicate data exactly the size of the given chunksize of the volume and on the position of multiples of the chunksize inside a single file. It does not give any storage space reduction if the data doesnt fit into these constraints. What concludes from this for practical usage is beyond the scope of this project. One would have to analyse it with various examples of typical data for different usage scenarios to say more to this point.

What really is the most important result from the project is that Opendedup seems to have quite some errors, or at least things that are not optimal. From what I've seen, I seriously doubt the safety of data stored on an Opendedup volume. So for this reason alone, I recommend not to use it in any application where data safety is a crucial issue.

List of references

Opendedup homepage:
www.opendedup.org

Source for various informations:
www.wikipedia.de

Virtual Box tutorial:
http://www.liberiangeek.net/2011/04/run-many-operating-systems-in-windows-with-virtualboxpart-one/

'du', 'ls -l' commands
http://linux.about.com/library/cmd/blcmdl1_du.htm
http://www.issociate.de/board/post/151527/The_du_and_ls_-l.html

Iozone
http://www.iozone.org/
http://www.iozone.org/docs/IOzone_msword_98.pdf

deduplication methods
http://recoverymonkey.org/2010/02/10/more-fud-busting-deduplication-%E2%80%93-is-variable-block-better-than-fixed-block-and-should-you-care/