- dd-Transfer -
Efficient Transfer of Large Hard Disk Images

1 Introduction

From time to time, the task of transferring the contents of an entire hard disk arises. This can happen, for example, to make a backup, to commission a new hard disk, or to transfer the contents of a storage medium to the cloud byte by byte.

Commercially available hard disks now have a capacity of up to ten terabytes and more. This volume of data is not easily transported over the existing bandwidth of standard Internet connections. It can even be a very tedious task for local networks.

This article first describes conventional ways and means to cope with the problem. From the indication of the limitations of these methods, the requirements and the partial steps to solve the problem are derived. The smal shell script ddtransfer.sh offers a solution for this points.

2 Conventional ways

For example, copying, transferring and importing the data of a USB stick with a size of 16 GB takes 68 minutes during normal working hours in Berlin over a connection of 100 MB, which is shared by more than 120 employees at the same time.

The individual steps were as follows:

1) Create the image

```
root@ddtransfer:~# time dd if=/dev/sdc bs=1G iflag=fullblock of=sdc.img
```

Duration: 12 minutes, 25 seconds

2) Transfer

```
root@ddtransfer:~# time rsync --compress --compress-level=9 sdc.img root@158.222.102.233:/
mnt/1/.
```

Duration: 51 minutes, 10 seconds

3) Importing the data into a storage volume

```
root@las-transfer2:/mnt/1# time dd if=sdc.img of=/dev/vdd bs=1G iflag=fullblock
```

Duration: 3 minutes, 28 seconds

I aborted a test with CloneZilla[1] as a cross-check, because I would have had to activate password authentication. Using Bar Bar Copy (bbcp[2]) would have speed up the transfer considerably, but the data would have been transferred unencrypted. On the other hand, the goal is to transfer very large

1 https://clonezilla.org/

2 http://www.slac.stanford.edu/~abh/bbcp/ und http://pcbunn.cithep.caltech.edu/bbcp/using_bbcp.htm

files. However, if images are created beforehand, an equally large amount of memory must be made available in advance (even in spite of compression). The Transferscript, on the other hand, does not require any extra free memory on the hard disk (apart from the need for a standard operating system).

Another way that customers are offered for the transfer of large amounts of data is to send hard disks, or to record a server (from about 10 TB), which is then sent back to the provider. Far beyond 10TB this option may be more practical, but with further technical development it is quite foreseeable that ddtransfer.sh will be a viable option even for extremely large amounts of data. A bandwidth of 1Gb/sec is no longer exceptional, the transfer of 10TB by a feasible task.

3 Requirements for a new program

With an image size of 15GB, the previously described procedure is completely normal and appropriate. The real problems only come into play when

a) significantly larger amounts of data are involved, which

b) must be transferred over a low bandwidth and

c) therefore the question of efficient data integrity checking also arises.

The duration of the transfer can quickly take several days. If a failure occurs during this time, the process of transfer must be restarted. The less data that has to be retransmitted, the less time is lost.

When the transfer of a partial step has been completed, it is possible to determine whether a bit flip or other changes have occurred by generating checksums after the transfer. With previous methods, this check can only be performed as an overall check; if a change is now detected, it is much more annoying at the end of a very long process.

The low bandwidth is the part that causes most of the processing time. However, very few programs are able to take full advantage of it. An effective way to use the bandwidth even more extensively is to call parallel connections. The respective connections are by no means processed at the same speed. It is not at all uncommon that when 16 connections are started in succession with an interval of a few seconds, the tenth transfer is completed first.

The script ddtransfer.sh is intended to close precisely these gaps.

1. The data to be transferred is divided into blocks and 'dd' is called for each of them to copy these several times in parallel.
2. A checksum is calculated for each block as it is created.
3. Eight parallel connections are used to send the blocks, depending on the bandwidth. This can be adjusted as required.
4. The blocks are written directly to the target device after transmission on the target system.
5. After writing each transmission to the target device, a checksum calculated from the completed processed block.

6. At the end, the checksums are compared. If there are any differences, the affected block is retransmitted and calculated.

As far as I know, no available tool for transferring large devices meets all these conditions. The program is based on 'dd', which belongs to the so-called core-utils of Unix, so it should run on every Linux that offers a shell, also the 'ssh-agent' should be available. So it can be used with a common live Linux, for example to transfer Windows installations. Likewise, it can be used on small hardware such as the Raspberry Pi, which serves as a transfer station.

The real sticking point of the whole procedure is the internet connection. Ddtransfer it is optimized for narrow bandwidths. Using the script also makes sense in local networks, in which case compression should be turned off, as well as in the case of a high-bandwidth Internet connection.

4 Issues

1 Disk space

Because ddtransfer.sh reads from the source device and writes to the destination device in parallel, there are no special requirements for available free disk space.

2 Bandwidth

If the program is called with a lot of connections, the throughput slows down for everyone else using the same network. This also means, as just described, that transferring at night time or on weekends will speed up the process significantly.

3 How dd works

'dd' can only either read or write. Because of this, it is advantageous to process reasonably large blocks in one operation. At the same time this program is able to address specific positions of a file or a block device bytewise. The exact position determination is essential for the division of the data to be transferred into clearly defined single steps. Therefore the block and file size is always a multiple of the block size of 512 bytes. So far I have allowed only one dd process per processor core (if there is only one core, there are two).

The way it works makes it somewhat awkward to determine whether dd is still writing to a device, or whether the process in question has already completed. Based on the size of a file, this may need to be calculated in advance and then checked on an ongoing basis. During the development of the script it therefore made sense to restart started write operations after an interruption. The option 'status=progress' is offered in new implementations of 'dd', but is not always available and not helpful.

5 Resumption after interruption

In case of a failure of the source or target computer the program can be called again. After its invocation ddtranfer.sh writes a report file. The name of the report file must be given as an option for the restart. This file contains the options of the previous call. Then the state of the data transfer and the created blocks are checked. Where necessary, interrupted sub-processes are restarted and the overall process is continued.

6 Closing

At the end, the report file is processed and checked whether the start and end checksums match or which ones are even missing. Missing final processing checksums can occur if the transfer has been interrupted. If the final checksum is missing or different, it is recalculated.

7 Commandline help

```
$ ./ddtransfer.sh

Usage: ./ddtransfer.sh -l|--local_device 'source_device' -r|--remote_device 'target_de-
vice' -H|--TargetHost 'target_host'
(--no_compression (by default compression is active)
(-p|--ssh_port 'ssh_port' (default is '22'))
(-m|--max_connections (max transmissions in parallel, default is 8))
(-S|--restart 'formerly written report file' (any other given option will be overwritten))
A running ssh-agent is necessary.

Tested with ext4, ntfs, xfs, btrfs.

It is recommended not to start any other dd execution on the local or target host while
this program is running.

For sudo users like Ubuntu, do:
1) 'sudo su -'
2) 'eval `ssh-agent -s`'
3) 'ssh-add /home/user/.ssh/id_rsa'
Now you can start the dd-transfer script.

Example:
./ddtransfer.sh -l /dev/vdd -r /dev/vdd -H 46.16.76.151
./ddtransfer.sh --local_device /dev/vdd --remote_device /dev/vdd --TargetHost
46.16.76.151

This transfers the contents from the local storage volume /dev/vdd to the remote volume
/dev/vdd on host 46.16.76.151.

Partitions like /dev/vdd1 are also usable.
```

8 Link

Public available source is https://github.com/ionos-enterprise/ionos-network-helper