dCache performance and its correlation to the pool size

Dr. Doris Ressmann and Dr. Silke Halstenberg
Karlsruher Institute of Technology, Steinbuch Centre for Computing,
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
E-mail: Doris.Ressmann@kit.edu; Silke.Halstenberg@kit.edu

Abstract. The size of hard disks is increasing every year, therefore it is natural that dCache [1] pool sizes are growing as well. The determination of the correct pool size is rather difficult. From a management point of view, it is preferable to only have a limited amount of pools. However, less and bigger pools aggravate operation. If one pool becomes unoperational, the complete storage behind this pool is offline. Additionally, restarting of bigger pools is time consuming and causes unnecessary delays. However, as a Tier-1 site like GridKa, which is serving all four LHC [2] experiments, the disk space provided behind a single server has to be optimised to the maximum one pool can provide. This paper analyses the coherence between pool size and the amount of transfer movers to result in highest throughput and less administrative impact.

1. Introduction
The current dCache production environment includes up to 12 pools with a maximum pool size of 16 TB behind one single file server. The decision to allocate only 16 TB is based on the fact that pools using control files have a restarting time of several hours. This long startup procedure causes lots of access failures for a long time. Once we had moved the meta data information into a Berkeley database, the startup time of a pool decreased. Still it was not clear if dCache can cope with very big pools.

Running many pools on the same host causes access problems, depending on the access patterns, since they do interfere with each other. Also tuning becomes very difficult: The parameters (e.g. amount of transfers running in parallel) can only be changed per pool not host wise. However, if other pools on this host are idle, it would be possible to allow more concurrent active transfers on one pool. Nevertheless, if all pools are very busy at the same time, the parameter has to be reduced for each of them separately. Currently this tuning can only be done manually, therefore having only one big pool on a host eases administration.

Looking at the history of the file sizes, it is evident that they are growing. As a result the amount of files per pool would not increase, if the pool size is increased. Table 1 indicates that especially for CMS and ATLAS the file size depends on the type of data. It seems that custodial nearline files are generally bigger than replica online files, which is a good starting point for improvement.

Also newer hardware becomes more and more powerful, which inspired us to start some test scenario evaluating different pool sizes with two different file sizes. Observing our current production environment 4 GB and 100 MB seem to be a good testing file size.
2. Test Environment

We ran several tests to write either big files (with a size of 4 GB each) or small files (100 MB each). Jobs are submitted via a Workload Management System (WMS) [3] to our workernodes, which generate a file and afterwards write it several times into the test dCache instance. For writing the SRM [4] protocol is used. In our test environment we use four different types of servers:

- On the headnode we operate the infoDomain, statisticsDomain, lmDomain, httpdDomain, dCacheDomain, adminDoorDomain, dirDomain, hoppingDomain, dcapDomain and the gPlazmaDomain. The host has a 64bit scientific linux with 2.6.9-89.0.11.ELsmp kernel and 4 GB memory.
- On the SRM node we run a PostgreSQL DB [5], utilityDomain, gsicapDomain and srmDomain. The host has a 64bit scientific linux with 2.6.9-42.0.10.ELsmp kernel and 4 GB memory.
- The services of the PNFS node run for the first tests on the headnode itself. But we realised that PNFS became the bottleneck. As a result we moved the corresponding PostgreSQL DB, PNFS itself as well as the pnfsManager to a new host with 64bit scientifc linux with 2.6.18-92.1.18.el5 kernel and 32 GB memory.
- The hosts for dCache pools and gridftp doors are all of the same type. They have a 64 bit scientific linux with 2.6.18-164.11.1.el5 kernel and 32 GB memory. However, for each test, all files are written to only one pool. Gridftp doors are only started on hosts which do not run an active pool during the corresponding test run. For each run there are a minimum of 2 gridftp doors active. And each pool host has 6 pools installed, with a size of 16, 32, 64, 96, 128 and 160 TB in a GPFS[6] cluster.

All involved servers have a 10G network connection. We use a GPFS[6] file system and tested with different block sizes. The first pools are installed with a block size of 1048576 Bytes. The other pools are installed with a block size of 4194304 Bytes and show a much better performance ratio as described in section 3. All pool servers are attached via two fibre channel connections. Both connections work in parallel so in theory it is possible to move up to 1600 MB per second in and out at the same time. Each logical disk consists of 10 physical (hardware) disks. Eight disks for data and two for parity (RAID6). How many logical disks a pool consists of can be seen in Table 2. This table also presents the test results of initial iozone tests, to illustrate the maximum write speed the disks are able to perform. The iozone command used is: "iozone -t 1 -i 0 -i 1 -s 45G -r 4M".

3. Observations and Optimization

Figures 1, 2 and 3 illustrate the file system throughput of 100 jobs each writing 100 files with a file size of 4 GB each. The jobs use srmcp to write the files into a 160 TB pool using 10, 25 and
Table 2. Pool setup parameters in the test environment

| pool               | blocksize | number of logical disks | iozone write MB/s |
|--------------------|-----------|-------------------------|-------------------|
| 16 TB BigFiles     | 1048576   | 1                       | 120               |
| 32 TB BigFiles     | 4194304   | 2                       | 440               |
| 64 TB BigFiles     | 4194304   | 4                       | 950               |
| 96 TB BigFiles     | 4194304   | 6                       | 1020              |
| 128 TB BigFiles    | 4194304   | 8                       | 1100              |
| 160 TB BigFiles    | 4194304   | 10                      | 1400              |
| 16 TB SmallFiles   | 4194304   | 1                       | 120               |
| 32 TB SmallFiles   | 4194304   | 2                       | 440               |
| 64 TB SmallFiles   | 4194304   | 4                       | 950               |
| 96 TB SmallFiles   | 4194304   | 6                       | 1020              |

30 active gridftp transfers at the same time. In dCache terms these transfers are usually called movers. For these tests we disabled checksum calculation, to be able to reach the maximum network throughput. Enabling the checksum should cause a lower write and a higher read rate for each file system. The figures clearly indicate that with a low as well as an excessive number of movers the maximum throughput cannot be reached. Too few movers cannot saturate the throughput and too many even cause a drastic behaviour, as all transfers are slowed down. They might cause transfers to run into a timeout and therefore have to be restarted, with a result that no transfer is completed. So it is very important to not allow too many gridftp transfers at the same time. Figure 4 characterises the interaction between transfer rate and the amount of transfers per host. Furthermore, the system load on a host with too many active movers might increase excessively, namely in such a way that the host becomes unresponsive.

![Figure 1](image1.png) System limitation not reached with 10 movers

![Figure 2](image2.png) Maximal network throughput reached with 25 movers

![Figure 3](image3.png) System saturated with 30 movers; each transfer is decelerated

Running the same tests with small files (100 MB each), the SRM negotiation became very noticeable. To be able to reach the disk throughput limit more than 100 concurrent movers had
to be active. However, already 100 active movers trigger a clear bottleneck in the PNFS database. As a result we positioned the PNFS services to a newer hardware, moving the bottleneck from PNFS into SRM. So getting the system to its optimal setup for small files, it is very important to have very powerful hardware for the headnodes.

In a real use case it can not be guaranteed that only small files are written to a particular pool, and having too many big file transfers active causes the system to be unresponsive. This and the limited amount of remaining time made us stop to continue tuning the system for small files.

3.1. Pool Startup Time
As bigger a pool is, as more files can be written to that pool. During startup each file in the pool repository is validated, which results in a longer starting time and a higher memory consumption (Table 3). However, even a 64 TB pool nearly filled with small files has an acceptable starting time of 379 seconds.

| pool           | number files | used disk in GB | start time in sec |
|----------------|--------------|-----------------|-------------------|
| 16 TB BigFiles | 7,237        | 15,618          | 4                 |
| 32 TB BigFiles | 7,000        | 27,996          | 5                 |
| 64 TB BigFiles | 14,899       | 63,135          | 16                |
| 96 TB BigFiles | 23,270       | 95,937          | 23                |
| 128 TB BigFiles | 31,983     | 127,928         | 32                |
| 160 TB BigFiles | 39,981      | 159,924         | 37                |
| 64 TB SmallFiles | 447,519   | 57,858          | 379               |

4. First Results
To decide about the pool size, the use case has to be determined. Large pools are manageable for big file sizes, since the amount of files in a pool is limited. On the contrary, storing small files into a very big pool is likely to cause trouble (e.g. unresponsiveness). Each file stored in a pool consumes a small amount of memory. Subsequently, a big pool filled with many small files
requires a large amount of memory. Furthermore, many files stored in one pool make this pool very attractive for read access, because a higher amount of files on a pool raises the probability of file access by jobs. To have the best disk throughput for small files, many concurrent transfers are required, the contrary is the case for big files. In this case only a limited amount of transfers are useful to get the best disk throughput.

Looking at our production environment, we see an occurrence of smaller files (200 to 300MB) for replica online spaces, which is a use case for smaller pool size. Pools with tape connection are usually filled with files with a slightly bigger file size (500 to 700MB) as can be seen in Table 1. Therefore the use of bigger pools is recommended. Furthermore, we have different pools in front of tape for reading and writing. In our configuration ATLAS and CMS do not read directly from those write pools. Usually all files are written via SRM, which means a constant stream to write a file from the beginning to the end. Clearly all files are only written once. If there is only one pool on a host, that pool can be configured much bigger. Pools with read access are more difficult to predict in their access pattern; different transfer protocols are used and many clients might access the same file at the same time. To be on the safe side it is much better to have smaller pools for such use cases. However, with growing file sizes, bigger pools can become more and more realistic even for pools which serve data to the client originally stored on tape or replica-online pools.

5. Future Prospects

It would be very interesting to analyse the bottlenecks of the headnodes in detail while transferring many small files in parallel. Also interesting is exploring a systems behavior, which is tuned exclusively for smaller files, while big files are written into this system. Certainly it is important to investigate the real use case meaning random file sizes are written to a pool.

Moreover, it would be very interesting to compare the SRM transfer tests described in this paper to other protocols like dcap or xrootd [7]. In addition, newer protocols like NFS 4.1 and consequently newer dCache versions should be compared as well.

6. Acknowledgements

Authors wishing to acknowledge assistance from colleagues Stephanie Boehringer, Verena Geisselmann, Dr. Christopher Jung, Xavier Mol and Jos van Wezel. Furthermore we want to thank Maria Dolores Prez Gutirrez from Data Direct Networks for her support.

References
[1] http://www.dcache.org
[2] http://lhcb.web.cern.ch/lhcb/
[3] http://glite.web.cern.ch/glite/wms/
[4] https://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.ws.op.doc
[5] http://www.postgresql.de/
[6] http://www-03.ibm.com/systems/software/gpfs/
[7] http://xrootd.slac.stanford.edu/