Analyzing data distribution on disk pools for dCache

S Halstenberg, C Jung and D Ressmann
Forschungszentrum Karlsruhe, Steinbuch Centre for Computing, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen

Abstract. Most Tier-1 centers of LHC Computing Grid are using dCache as their storage system. dCache uses a cost model incorporating CPU and space costs for the distribution of data on its disk pools. Storage resources at Tier-1 centers are usually upgraded once or twice a year according to given milestones. One of the effects of this procedure is the accumulation of heterogeneous hardware resources. For a dCache system, a heterogeneous set of disk pools complicates the process of weighting CPU and space costs for an efficient distribution of data. In order to evaluate the data distribution on the disk pools, the distribution is simulated in Java. The results are discussed and suggestions for improving the weight scheme are given.

1. Introduction

dCache [1] is a data management software for storing and retrieving huge amounts of data. It is capable of distributing data among heterogenous servers. As it is a storage element in the gLite middleware [2] and is used at many WLCG grid sites, it supports multiple access protocols, e.g. GridFTP [3] and SRM [4]. dCache uses a single virtual file system tree. It can run with or without a tape back-end.

All WLCG Tier-1 centers have a tape back-end; this also applies to most of the larger Tier-2 centers. In order to describe properties of a dCache system at a Tier-1 center, some properties of the dCache system and its hardware at GridKa Computing Centre [5] at Forschungszentrum Karlsruhe are summarized in the following (numbers as of middle of March, 2009):

- supports all four LHC VOs and additional 20 VOs,
- 300 logical dCache pools on 100 physical machines with GPFS [6],
- 1.2 PB of disk space as tape cache and 1 PB disk-only space,
- the tape system uses the Tivoli Storage Manager [7], there are two tape libraries which currently are storing 2 PB of dCache data,
- the storage space for the WLCG VOs is upgraded regularly.

Due to the latter, the hardware used is heterogenous; when new disks are made available to the system, the data distribution is also inhomogeneous.

2. Cost calculation in dCache
dCache distributes data based on a cost model, i.e. by calculating a cost for the transfer for each pool which is eligible for receiving the file and then choosing the pool with the lowest cost. There are two different kinds of costs: space cost and performance cost. Two schemes for cost calculation have been implemented in dCache; in the following only the new scheme, as described in dCache book [1], will be evaluated and discussed.
2.1. Performance cost
At first, a performance cost is calculated for each kind of transfer mode available on the pool. The performance cost depends on the number of transfers to/from the pool (including the transfers which are queued) and the maximum number of allowed transfers:

$$\text{perfCost}(\text{transferMode}) = \frac{\# \text{activeTransfers} + \# \text{waitingTransfers}}{\# \text{maxAllowed}}.$$ 

The overall performance cost is the average of the performance costs for the allowed transfer modes on the pool:

$$\text{perfCost} = \frac{\sum_{\text{transferModes}} \text{perfCost}(\text{transferMode})}{\# \text{transferModes}}.$$ 

2.2. Space cost
There are two parameters which are used for the space cost: the gap parameter describes the minimum of free space on a pool so that the pool can be considered not (nearly) full; the breakeven parameter $\in [0, 1]$ is a linear factor which is used if the pool is considered nearly full.

If a pool is not nearly full, the space cost is proportional to the size of the file which will be transferred and inversely proportional to the free space of the pool. Otherwise the space cost is dependent on the age of the least recently used file, lruAge, only.

$$\text{spaceCost} = \begin{cases} 3 \cdot \frac{\text{newFileSize}}{\text{freeSpace}}, & \text{if freeSpace} > \text{gap}; \\ 1 + \text{breakeven} \cdot 7 \cdot 24 \cdot 60, & \text{if freeSpace} < \text{gap} \text{ and lruAge} < 60s; \\ 1 + \frac{\text{breakeven} \cdot 7 \cdot 24 \cdot 60 \cdot 60s}{\text{lruAge}}, & \text{if freeSpace} < \text{gap} \text{ and lruAge} > 60s. \end{cases}$$

This formula is not intuitively conceivable (e.g. the factor of 3 for freeSpace > gap is a fixed parameter); a visualized example is given Figure 1.

The overall cost is a simple linear combination of the space cost and of the performance cost, with the administrator choosing the space cost factor, scf, and the performance cost factor, pcf.

$$\text{overallCost} = \text{scf} \cdot \text{spaceCost} + \text{pcf} \cdot \text{perfCost}.$$ 

3. Simulation of costs
At first, file transfers with a high rate to one pool only will be simulated using one cost scheme, i.e. either the space cost factor or the performance cost factor is set to zero. This also helps to visualize the weight functions.

In the second subsection, the costs will be simulated for a set up in which there are two nearly full pools and one empty pool. This represents the scenario of a hardware upgrade/addition.

The simulation has been implemented in Java [8].

---

1 There are two modes for HSM transfers (store and restore), one mode for transfers from and to clients (client) and two modes for internal pool-to-pool-transfers (pool-to-pool server and pool-to-pool client). Not all modes need to be enabled on a pool.
3.1. Using either space costs or performance costs only

In this subsection, we will assume one pool with a size of 2 TB, which will be filled with 1000 files of file size 2 GB each. Every 10 seconds, a new transfer from the client will be initiated. All transfers share an effective bandwidth of 100 MB/s. The only transfer mode allowed for the pool is the client transfer. The breakeven parameter has been set to 0.9 and maximum number of client transfers allowed, \( \# \text{maxAllowedClient} \), is 10.

At first we consider space costs only, i.e. \( scf = 1.0 \) and \( pcf = 0.0 \). The graph of the cost function and the graph of the free space available are given in Figure 1. As the bandwidth for transfers is a fixed value, the free space declines linearly until the pool is full. The behavior of the space cost is more interesting. It is below 0 until about 80% of the space is in use. Getting close to the gap value, the space cost rises more than exponentially and jumps at the gap value. When all transfers have finished, the space cost declines hyperbolically. The jump at the gap value is inversely proportional to \( \text{truAge} \). In our example the least recently used file was transferred when the simulation began; therefore the jump at gap value is about two orders of magnitude.

Using performance cost only, i.e. \( scf = 0.0 \) and \( pcf = 1.0 \), the graph of the costs, which is given in Figure 2, has a different shape. As transfers are being queued, the cost rises basically linearly until all transfers have been submitted, then it declines basically linearly.

3.2. Scenario of adding one free pool using the standard cost model

As mentioned in the introduction, storage space at Tier-1s is regularly expanded. A scenario based on such an expansion is discussed in this subsection.

Of course administrators do not use the space or the performance cost exclusively, but a linear combination of them. This is used for the simulation of adding one pool to a set of

---

2 The decimal definition is used, i.e. 1 TB = \( 10^{12} \) B.
3 In practice, the workloads are more complex, of course. This scenario has been taken for two reasons: it can be understood intuitively and it is nevertheless a good approximation for real workloads.
4 The minor jumps in Figure 2 are due to (sometimes multiple) transfers finishing.
already nearly filled pools. The most simple formula,

\[ totalCost = spaceCost + perfCost, \]

(i.e. \( scf = 1.0 \) and \( ccf = 1.0 \)) is applied.

The scenario simulated is using the following set of parameters: we have three pools of 2 TB each. All the files have a file size of 2 GB. At the beginning of the simulation, 95% of the disk space of pool number one are used, while for pool number two 90% is used; data on both pools is in the status cached, i.e. it can be deleted if space is needed. For sake of simplicity, all files have a \( lruAge \) of 6 hours. Pool number three is completely empty.

All pools have the following set of parameters in common: a \( breakeven \) parameter of 0.9, a \( breakeven \) parameter of 10 GB and a maximum number of concurrent client transfers of 10. As in previous simulations, only one queue, the client queue, is considered and used in calculating the pool costs. The result of the simulation is shown in Figure 3.

As expected, the empty pool is the last one to reach a high cost. After about 2000 s, all three pools have similar costs. This results in cached files being removed from pool one and two; overall about 850 GB are removed from these pools, i.e. about 35% of the data transferred in this simulation.

After the costs for all three pools reach a joint local maximum after 5000 s, the costs decline differently. For pool number one and two, the overall cost is dominated by the space cost, while for pool three the space cost is negligible and the performance cost declines as the transfer queue is being emptied.

Of course, a high number of pools simplifies the distribution of file transfers. This is often not possible as partitioning of the overall disk space is often needed, e.g. because of different experiments using one dCache instance.

4. Summary and suggestions
The dCache storage system chooses logical pools for data transfer based on a cost system. This system incorporates free disk space (‘space cost’) and the number of incoming and outgoing transfers to the pool (‘performance cost’).
Figure 3. Both plots depict the result of the simulation described in subsection 3.2. In left-hand graphic, the free space on the three pools is shown. Exactly 2 TB of data are transferred to the pools (who have total free space of 2.15 TB), yet there is nearly 1 TB of free space on pool number three, i.e. about 0.85 TB of cached data has been deleted from pools one and two. The right-hand graphic depicts the overall pool cost for each of the pools; depending on the free space in the beginning, they reach the maximal pool cost on different points of time.

Both costs follow intuitive concepts. A small number of transfers leads to a domination of the space cost, while a high number of transfers leads to a domination of the performance cost (unless pools are basically full). The challenging task is combining these costs, as dCache sites face different use cases.

A simple, yet a very common, scenario, the addition of hardware, was simulated in subsection 3.2. This already showed the complex behavior of the overall cost and therefore the choice of the pool for the transfer. It basically seems impossible to find a linear combination of the costs which accommodates all use cases, as there are various use cases and setups.

Therefore the authors of this paper suggest to change the way space cost and performance cost are calculated. There are several possibilities:

- smoothing the space cost function at the gap value,
- not using free space but free and removable space in calculating space cost, while giving a weight factor to the removable space,
- changing the linear growth for performance cost to an exponential growth as soon as the maximum number of concurrent transfers has been reached,
- weighting the files in the queues by their file sizes for calculating the performance cost, and
- replacing \textit{lruAge} by an average age of the oldest files.

In addition, the cost model could be replaced by a probability model, i.e. dCache could calculate a probability for a file to be transferred to a pool and then randomly choose the pool based on these probabilities. This would spread the transfers better over the pools.

Acknowledgments
This paper would not have been possible without the support of the dCache developers. They provided detailed explanation on the current cost model; we also want thank them for year long collaboration.

We also would like to express our thanks to Xavier Mol for fruitful discussions on the topic of this paper.
References
[1] dCache: http://www.dcache.org
[2] gLite: http://glite.web.cern.ch/glite/
[3] GT 4.0 GridFTP: http://www.globus.org/toolkit/docs/4.0/data/gridftp/
[4] Storage Resource Manager Working Group: http://sdm.lbl.gov/srm-wg/index.html
[5] Grid Computing Centre Karlsruhe: http://www.gridka.de
[6] IBM General Parallel File System: http://www-03.ibm.com/systems/clusters/software/gpfs/index.html
[7] Tivoli Storage Manager: http://www-01.ibm.com/software/tivoli/products/storage-mgr/
[8] Java: http://www.java.com