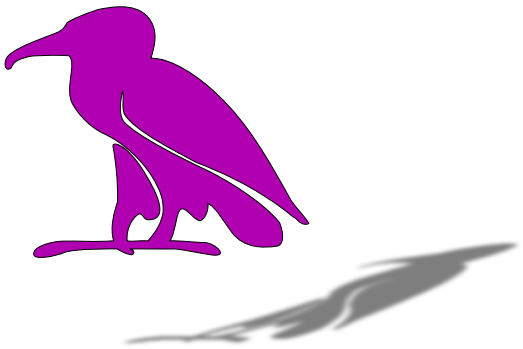
The dCache Book

for 1.9.5-series



The dCache Book: for 1.9.5-series

Abstract

The dCache Book is the guide for administrators of dCache systems. The first part describes the installation of a simple single-host dCache instance. The second part describes the components of dCache and in what ways they can be configured. This is the place for finding information about the role and functionality of components in dCache as needed by an administrator. The third part contains solutions for several problems and tasks which might occur during operating of a dCache system. Finally, the last two parts contain a glossary and a parameter and command reference.

Table of Contents

Part I. Getting started

Table of Contents

In this part is intended for people who are new to dCache. It gives an introduction to dCache, including how configure a simple setup, and details some simple and routine administrative operations.

Chapter 1. Introduction

dCache is a distributed storage solution. It organises storage across computers so the combined storage can be used without the end-users being aware of on precisely which computer their data is stored; end-users see simply a large amount of storage.

Because end-users need not know on which computer their data is stored, their data can be migrated from one computer to another without any interruption of service. This allows dCache storage computers to be taken out of service or additional machines (with additional storage) to be added without interrupting the service the end-users enjoy.

dCache supports requesting data from a tertiary storage system. A tertiary storage system typically uses a robotic tape system, where data is stored on a tape from a library of available tapes, which must be loaded and unloaded using a tape robot. Tertiary storage systems typically have a higher initial cost, but can be extended cheaply by added additional tapes. This results in tertiary storage systems being popular where large amounts of data must be read.

dCache also provides many transfer protocols (allowing users to read and write to data). These have a modular deployment, allowing dCache to support expanded capacity by providing additional front-end machines.

Another performance feature of dCache is hot-spot data migration. In this process, dCache will detect when a few file are being requested very often. If this happens, dCache can make duplicate copies of the popular files on other computers. This allows the load to be spread across multiple machines, so increasing throughput.

The flow of data within dCache can also be carefully controlled. This is especially important for large sites as chaotic movement of data may lead to suboptimal usage; instead, incoming and outgoing data can be marshaled so they use designated resources; allowing better throughput and guaranteeing end-user experience.

dCache provides a comprehensive administrative interface for configuring the dCache instance. This is described in the later sections of this book.

Cells and Domains

dCache, as distributed storage software, can provide a coherent service using multiple computers or *nodes* (the two terms are used interchangeable). Although dCache can provide a complete storage solution on a single computer, one of its strengths is the ability to scale by spreading the work over multiple nodes.

A *cell* is dCache's most fundamental executable building block. Even a small dCache deployment will have many cells running. Each cell has a specific task to perform and most will interact with other cells to achieve it.

Cells can be grouped into common types; for example, pools, doors. Cells of the same type behave in a similar fashion and have higher-level behaviour (such as storing files, making files available). Later chapters will describe these different cell types and how they interact in more detail.

There are only a few cells where (at most) only a single instance is required. The majority of cells within a dCache instance can have multiple instances and dCache is designed to allow load-balancing over these cells.

A *domain* is a container for running cells. Each domain runs in its own Java Virtual Machine (JVM) instance, which it cannot share with any other domain. In essence, a domain *is* a JVM with the additional functionality

necessary to run cells (such as system administration and inter-cell communication). Since domains running on the same node and each have an independent JVM, they share the node's resources such as memory, available CPU and network bandwidth.

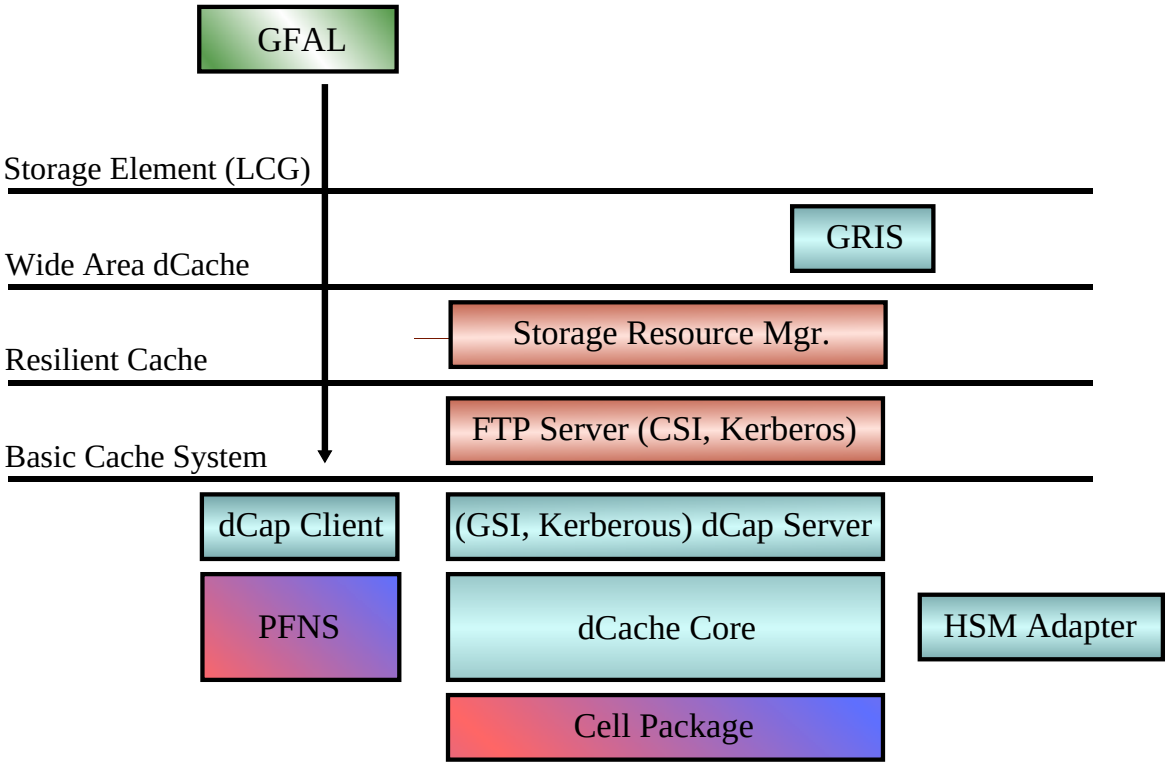
dCache comes with a set of domain definitions, each specifying a useful set of cells to run within that domain to achieve a certain goal. These goals include storing data, providing a front-end to the storage, recording filenames, and so on. The list of cells to run within these domains are recommended deployments: the vast majority of dCache deployments do not need to alter these lists.

A node is free to run multiple domains, provided there's no conflicting requirement from the domains for exclusive access to hardware. A node may run a single domain; but, typically a node will run multiple domains. The choice of which domains to run on which nodes will depend on expected load of the dCache instance and on the available hardware. If this sounds daunting, don't worry: starting and stopping a domain is easy and migrating a domain from one node to another is often as easy as stopping the domain on one node and starting it on another.

dCache is scalable storage software. This means that (in most cases) the performance of dCache can be improved by introducing new hardware. Depending on the performance issue, the new hardware may be used by hosting a domain migrated from a overloaded node, or by running an additional instance of a domain to allow load-balancing.

Most cells communicate in such a way that they don't rely on in which domain they are running. This allows a site to move cells from one domain to another or to create new domain definitions with some subset of available cells. Although this is possible, it is rare that redefining domains or defining new domains is necessary. Starting or stopping domains is usually sufficient for managing load.

Figure 1.1. The dCache Layer Model



The layer model shown in Figure 1.1, “The dCache Layer Model” gives an overview of the architecture of the dCache system.

Chapter 2. Installing dCache

Michael Ernst

Patrick Fuhrmann

Mathias de Riese

The first section describes the installation of a fresh dCache instance using RPM files downloaded from the dCache home-page [http://www.dcache.org]. It is followed by a guide to upgrading an existing installation. In both cases we assume standard requirements of a small to medium sized dCache instance without an attached *tertiary storage system*. The third section contains some pointers on extended features.

Installing a Single Node dCache Instance

In the following the installation of a single node dCache instance will be described. The Chimera name space provider, some management components, and the SRM need a PostgreSQL server installed. We recommend running this PostgreSQL on the local node. The first section describes the configuration of a PostgreSQL server. After that the installation of Chimera and of the dCache components will follow. During the whole installation process root access is required.

Prerequisites

In order to install dCache the following requirements must be met:

- An RPM-based Linux distribution is required for the following procedure. For Debian derived systems the RPM may be converted to a DEB using alien. Solaris is supported using either the Solaris package or the tarball.
- dCache 1.9 requires Java 1.5 or 1.6 SDK. We recommend Java 1.6. It is recommended to use the newest Java release available within the release series used.
- PostgreSQL must be installed and running. See the section called “Installing a PostgreSQL Server” for more details. It is strongly recommended to use version 8 or higher.

Installation of the dCache Software

The RPM packages may be installed right away, for example using the command:

```
[root] # rpm -ivh dcache-server-version-release.i386.rpm
[root] # rpm -ivh dcache-client-version-release.i386.rpm
```

The actual sources lie at http://www.dcache.org/downloads.shtml. To install for example Version 1.9.4-2 of the server you would use this:

```
[root] # rpm -ivh http://www.dcache.org/downloads/1.9/dcache-server-1.9.4-2.noarch.rpm
```

The Client can be found in the download-section of the above url, too.

Readying the PostgreSQL server

You must configure PostgreSQL for use by dCache and create the necessary PostgreSQL user accounts and database structure. This section describes how to do this.

Configuring the PostgreSQL server

Using a PostgreSQL server with dCache places a number of requirements on the database. This section describes what configuration is necessary to ensure PostgreSQL operates so dCache can use it.

Restarting PostgreSQL

If you have edited PostgreSQL configuration files, you *must* restart PostgreSQL for those changes to take effect. On many systems, this can be done with the following command:

```
[root] # /etc/init.d/postgresql restart
```

Enabling TCP connections

Important

For Versions of PostgreSQL newer than 8.0 the TCP connections are already enabled and this section has to be ignored.

When connecting to PostgreSQL, dCache will always use TCP connections. So, for dCache to use PostgreSQL, support for TCP sockets must be enabled. We realize UNIX domain sockets are easier to work with from a security point of view, however there is no way to use UNIX domain sockets from a Java application.

In contrast to dCache, the PostgreSQL stand-alone client application **psql** can connect using either a TCP socket or via a UNIX domain socket. Because of this, it is common for PostgreSQL to disable TCP sockets by default, requiring the admin to explicitly configure PostgreSQL so connecting via a TCP socket is supported.

To enable TCP sockets, edit the PostgreSQL configuration file `postgresql.conf`. This is often found in the `/var/lib/pgsql/data`, but may be located elsewhere. You should ensure that the line `tcpip_socket` is set to `true`; for example:

```
tcpip_socket = true
```

Enabling local trust

Perhaps the simplest configuration is to allow password-less access to the database and the following documentation assumes this is so.

To allow local users to access PostgreSQL without requiring a password, ensure the file `pg_hba.conf`, usually located in `/var/lib/pgsql/data`, contains the following lines.

```
local   all         all                         trust
host    all         all         127.0.0.1/32    trust
host    all         all         ::1/128         trust
```

Note

Please note it is also possible to run dCache with all PostgreSQL accounts requiring passwords.

Configuring Chimera

Chimera is a library providing a hierarhical name space with associated meta data. Where pools in dCache store the content of files, Chimera stores the names and meta data of those files. Chimera itself stores the data in a relational database. We will use PostgreSQL in this tutorial.

Note

dCache used to use another name space implementation called `pnfs`. `pnfs` is still available, we do however recommend that new installations use Chimera.

Initialize the database

Create the Chimera user and database and add the Chimera-specific tables and stored procedures:

```
[root] # createdb -U postgres chimera
CREATE DATABASE

[root] # createuser -U postgres --no-superuser --no-createrole --createdb --pwprompt chimera
Enter password for new role:
Enter it again:
CREATE ROLE

[root] # psql -U chimera chimera -f /opt/d-cache/libexec/chimera/sql/create.sql
psql:/opt/d-cache/libexec/chimera/sql/create.sql:23: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_inodes_pkey" for table "t_inodes"
CREATE TABLE
psql:/opt/d-cache/libexec/chimera/sql/create.sql:35: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_dirs_pkey" for table "t_dirs"
CREATE TABLE
psql:/opt/d-cache/libexec/chimera/sql/create.sql:45: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_inodes_data_pkey" for table "t_inodes_data"
many more like this...
INSERT 0 1
many more like this...
INSERT 0 1
CREATE INDEX
CREATE INDEX
psql:/opt/d-cache/libexec/chimera/sql/create.sql:256: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_storageinfo_pkey" for table "t_storageinfo"
CREATE TABLE
psql:/opt/d-cache/libexec/chimera/sql/create.sql:263: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_access_latency_pkey" for table "t_access_latency"
CREATE TABLE
psql:/opt/d-cache/libexec/chimera/sql/create.sql:270: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_retention_policy_pkey" for table "t_retention_policy"
CREATE TABLE
psql:/opt/d-cache/libexec/chimera/sql/create.sql:295: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_locationinfo_pkey" for table "t_locationinfo"
CREATE TABLE
psql:/opt/d-cache/libexec/chimera/sql/create.sql:311: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_locationinfo_trash_pkey" for table "t_locationinfo_trash"
CREATE TABLE
CREATE INDEX
psql:/opt/d-cache/libexec/chimera/sql/create.sql:332: NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index "t_acl_pkey" for table "t_acl"
CREATE TABLE
CREATE INDEX
```

```
[root] # createlang -U postgres plpgsql chimera
[root] # psql -U chimera chimera -f /opt/d-cache/libexec/chimera/sql/pgsql-procedures.sql
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE TRIGGER
CREATE FUNCTION
CREATE TRIGGER
CREATE SEQUENCE
CREATE FUNCTION
CREATE TRIGGER
```

Database connection settings can be customized in `/opt/d-cache/config/chimera-config.xml` Specifically you should change the user to “chimera”.

```
<?xml version="1.0"?>
<config>
        <db fsid="0" url="jdbc:postgresql://localhost/chimera?prepareThreshold=3" drv="org.postgresql
.Driver" user="chimera" pass="" dialect="PgSQL" />
        <nfs>
                <port>2049</port>
                <logLevel>0</logLevel>
                <logFile>/tmp/himera.log</logFile>
        </nfs>
</config>
```

Mounting Chimera through NFS

Although most components in dCache access the Chimera database directly, some rely on a mounted file system for access. The mounted file system is also nice for administrative access. This offers the opportunity to use OS-level tools like ls and mkdir for Chimera. However, direct I/O-operations like cp are not possible, since the NFSV3 interface provides the namespace part only. This section describes how to start the Chimera NFS3 server and mount the name space.

Chimera NFS server uses `/etc/exports` file to manage exports. So it has to exist or be created. The typical `exports` file looks like this:

```
/ localhost(rw)
/pnfs
# or
# /pnfs *.my.domain(rw)
```

Since Chimera is coupled with dCache it uses the same configuration file and won’t start without it. So copy the `/opt/d-cache/etc/dCacheSetup.template` to `/opt/d-cache/config/dCacheSetup`.

Note

On some linux distributions you might have to switch the portmap daemon off before starting chimera:

```
[root] # /etc/init.d/portmap stop
Stopping portmap: portmap
```

Start it via script:

```
[root] # /opt/d-cache/libexec/chimera/chimera-nfs-run.sh start
```

To automate the launching of that script at startup time, link to it from `/etc/init.d/`. Then announce it to chkconfig:

```
[root] # chkconfig --add chimera-nfs-run.sh
[root] # chkconfig chimera-nfs-run.sh on
```

First we create the root of the Chimera namespace, called 'pnfs' for legacy reasons.

```
[root] # /opt/d-cache/libexec/chimera/chimera-cli.sh Mkdir /pnfs
```

Now we need to add directory tags. For more information on tags see the section called "Directory Tags".:

```
[root] # /opt/d-cache/libexec/chimera/chimera-cli.sh Mkdir /pnfs/your domain
[root] # /opt/d-cache/libexec/chimera/chimera-cli.sh Mkdir /pnfs/your domain/data
[root] # echo "chimera" | /opt/d-cache/libexec/chimera/chimera-cli.sh Writetag /pnfs/your domain
/data sGroup
[root] # echo "StoreName sql" | /opt/d-cache/libexec/chimera/chimera-cli.sh Writetag /pnfs/your do
main/data OSMTemplate
```

If you plan to use `dCap` with mounted file system instead of the URL-syntax (e.g. **dccp** `/pnfs/desy.de/data/file1` `/tmp/file1`), we need to mount the root of Chimera locally (remote mounts are not allowed yet). This will allow us to establish wormhole files so `dCap` clients can discover the `dCap` doors.

```
[root] # mount localhost:/ /mnt
[root] # mkdir /mnt/admin/etc/config/dCache
[root] # touch /mnt/admin/etc/config/dCache/dcache.conf
[root] # touch /mnt/admin/etc/config/dCache/'.(fset)(dcache.conf)(io)(on)'
[root] # echo "door host:port" > /mnt/admin/etc/config/dCache/dcache.conf
```

The default values for ports can be found in Chapter 31, *dCache default port values*. They can be altered in `/opt/d-cache/config/dCacheSetup`

The configuration is done now, so unmount Chimera:

```
[root] # umount /mnt
```

Please note that whenever you need to change the configuration, you have to remount the root `localhost:/` to a temporary location like `/mnt`.

The "user's view" of Chimera is automatically mounted by the dCache init script. You have to make sure that the mountpoint is created on the machine (`/pnfs`). Chimera can be mounted manually with:

```
[root] # mkdir /pnfs
[root] # mount localhost:/pnfs /pnfs
```

Creating users and databases for dCache

The dCache components will access the database server with the user srmdcache which can be created with the **createuser**; for example:

```
[root] # createuser -U postgres --no-superuser --no-createrole --createdb --pwprompt srmdcache
```

Several management components running on the head node as well as the SRM will use the database dcache for state information:

```
[root] # createdb -U srmdcache dcache
```

There might be several of these on several hosts. Each is used by the dCache components running on the respective host.

```
[root] # createdb -U srmdcache companion
[root] # psql -U srmdcache companion -f /opt/d-cache/etc/psql_install_companion.sql
```

If the resilience feature provided by the *replica manager* is used, the database "replicas" has to be prepared on the head node with the command:

```
[root] # createdb -U srmdcache replicas
[root] # psql -U srmdcache replicas -f /opt/d-cache/etc/psql_install_replicas.sql
```

Note

Note that the disk space will at least be cut in half if the replica manager is used.

If the billing information should also be stored in a database (in addition to files) the database billing has to be created:

```
[root] # createdb -U srmdcache billing
```

However, we strongly advise against using the same database server for Chimera and the billing information. For how to configure the `billing` to write into this database, see below.

Installing dCache Components

The main configuration file of a dCache instance is `/opt/d-cache/config/dCacheSetup`. Set the variable `java` to the binary of the Java VM and the variable `serviceLocatorHost` to the hostname of the single node running dCache.

Use the templates of the configuration files found in `/opt/d-cache/etc/` to create the following files.

The installation and start-up scripts use the information in `/opt/d-cache/etc/node_config`. First copy it from the template. For a setup with a single node, set `NODE_TYPE` to "admin". To enable doors on this node, add the respective doors to `SERVICES`, for instance "dcap" or "gridftp". Set `NAMESPACE` to "chimera".

For authorization of grid users the file `/opt/d-cache/etc/dcache.kpwd` is needed. You can simply copy the template that is in the same directory. Note that it may be generated from the standard `/etc/`

`grid-security/grid-mapfile` with the tool `grid-mapfile2dcache-kpwd` which is distributed with the WLCG software.

We proceed by finalising the initial configuration by executing **/opt/d-cache/install/install.sh**, for example:

```
[root] # /opt/d-cache/install/install.sh
INFO:Skipping ssh key generation

 Checking MasterSetup  ./config/dCacheSetup O.k.

   Sanning dCache batch files

    Processing adminDoor
    Processing chimera
    Processing dCache
    Processing dir
    Processing door
    Processing gPlazma
    Processing gridftpdoor
    Processing gsidcapdoor
    Processing httpd
    Processing info
    Processing infoProvider
    Processing lm
    Processing maintenance
    Processing chimera
    Processing pool
    Processing replica
    Processing srm
    Processing statistics
    Processing utility
    Processing xrootdDoor


 Checking Users database .... Ok
 Checking Security       .... Ok
 Checking JVM ........ Ok
 Checking Cells ...... Ok
 dCacheVersion ....... Version production-1-9-3-1
```

No pools have been created on the node yet. Adding pools to a node is a two step process:

1. The directory layout of the pool is created and filled with a skeleton configuration using **dcache pool create *poolSize poolDirectory***, where *poolDirectory* is the full path to the directory which will contain the data files as well as some of the configuration of the pool, and *poolSize* is the size of the pool, specified in bytes or with a M, G, or T suffix (for mibibytes, gibibytes and tibibytes, respectively).

   Make sure that there is always enough space under *poolDirectory*. Be aware that only pure data content is counted by dCache. Leave enough room for configuration files and filesystem overhead.

   Creating a pool does not modify the dCache configuration.

2. The pool is given a unique name and added to the dCache configuration using **dcache pool add *poolName poolDirectory***, where *poolDirectory* is the directory in which the pool was created and *poolName* is a name for the pool. The name must be unique throughout the whole dCache installation, not just on the node.

   Adding a pool to a configuration does not modify the pool or the data in it and can thus safely be undone or repeated.

Note

The default gap for poolsizes is 4GiB. This means you should make a bigger pool than 4GiB otherwise you would have to change this gap in the dCache admin tool. See the example below. See also the section called “The Admin Interface”.

```
(local) admin > cd poolName
(poolname) admin > set gap 2G
(poolname) admin > save
```

An example may help to clarify the use of these commands:

```
[root] # /opt/d-cache/bin/dcache pool create 500G /q/pool1
Created a 500 GiB pool in /q/pool1. The pool cannot be used until it has
been added to a domain. Use 'pool add' to do so.

Please note that this script does not set the owner of the pool directory.
You may need to adjust it.
[root] # /opt/d-cache/bin/dcache pool add myFirstPool /q/pool1/

Added pool myFirstPool in /q/pool1 to dcache-vmDomain.

The pool will not be operational until the domain has been started. Use
'start dcache-vmDomain' to start the pool domain.
[user] $ /opt/d-cache/bin/dcache pool ls
Pool        Domain                       Size   Free Path
myFirstPool dcache-vmDomain               500    550 /q/pool1
Disk space is measured in GiB.
```

All configured components can now be starting with **`dcache start`**, for example:

```
[root] # /opt/d-cache/bin/dcache start
Starting lmDomain  Done (pid=7514)
Starting dCacheDomain  Done (pid=7574)
Starting pnfsDomain  Done (pid=7647)
Starting dirDomain  Done (pid=7709)
Starting adminDomain  Done (pid=7791)
Starting httpdDomain  Done (pid=7849)
Starting utilityDomain  Done (pid=7925)
Starting gPlazma-dcache-vmDomain  Done (pid=8002)
Starting infoProviderDomain  Done (pid=8081)
Starting dcap-dcache-vmDomain  Done (pid=8154)
Starting gridftp-dcache-vmDomain  Done (pid=8221)
Starting gsidcap-dcache-vmDomain  Done (pid=8296)
Starting dcache-vmDomain  Done (pid=8369)
```

Installing a Multi Node dCache Instance

The previous section decsribed how to install a single node dCache installation. A typically medium-sized dCache installation will however have a single head node hosting the name space provider and other central components, and a number of pool nodes. It is common to also use pool nodes as FTP and DCAP doors.

The Chimera file system must be mounted on all nodes running either the `SRM` or `GridFTP`. Client nodes relying on `dCap` access without using URLs also need to mount Chimera. Pools do not need a mount anymore. Having a mount on the Chimera/NFS3-server node itself is always a good idea as it eases maintenance.

To mount the Chimera file system, either modify `config/chimera-config.xml` such that it points towards the correct PostgreSQL host and start a local Chimera NFSv3 server locally, or mount the NFS file system exported from the head node. In the latter case, set `NAMESPACE_NODE` in `etc/node_config` to the host running the Chimera NFSv3 server.

For the head node, follow the description of the previous chapter, but do not create any pools. For pools and for `dCap` or `GridFTP` doors, PostgreSQL is not needed and installation of PostgreSQL can be skipped on nodes that only hosts these services. Proceed by creating `config/dCacheSetup`; `serviceLocatorHost` has to be set to the name of the head node. In `etc/node_config` leave `NODE_TYPE` empty. Add any doors you want to start on this node to `SERVICES` and set `NAMESPAE` to “chimera”. Run `install/install.sh` to finish the installation. Finally, use **dcache pool create** and **dcache pool add** to create and add pools on this node.

Upgrading a dCache Instance

Upgrading to bugfix releases within one version (e.g. from 1.9.3-1 to 1.9.3-2) may be done by shutting down the server and upgrading the packages with

```
[root] # rpm -Uvh packageName
```

Follow this by rerunning **install/install.sh**. For details on the changes, please refer to the change log on the download page.

Chapter 3. Getting in Touch with dCache

This section is a guide for exploring a newly installed dCache system. The confidence obtained by this exploration will prove very helpful when encountering problems in the running system. This forms the basis for the more detailed stuff in the later parts of this book. The starting point is a fresh installation according to the the section called “Installing a Single Node dCache Instance”.

Checking the Functionality

First, we will get used to the client tools. On the dCache head node, change into the `pnfs` directory, where the users are going to store their data:

```
[user] $ cd /pnfs/site.de/data/
[user] $
```

The mounted Chimera filesystem is not intended for reading or writing actual data with regular file operations via the NFS protocol.

Reading and writing data to and from a dCache instance can be done with a number of protocols. After a standard installation, these protocols are `dCap`, `GSIdCap`, and `GridFTP`. In addition dCache comes with an implementation of the `SRM` protocol which negotiates the actual data transfer protocol.

We will first try `dCap` with the **dccp** command:

```
[user] $ export PATH=/opt/d-cache/dcap/bin/:$PATH
[user] $ cd /pnfs/site.de/data/
[user] $ dccp /bin/sh my-test-file
541096 bytes in 0 seconds
```

This command succeeds if the user `user` has the Unix rights to write to the current directory `/pnfs/site.de/data/`.

The dccp command also accepts URLs. We can copy the data back using the dccp command and the `dCap` protocol but this time describing the location of the file using a URL.

```
[user] $ dccp dcap://adminNode/pnfs/site.de/data/my-test-file /tmp/test.tmp
541096 bytes in 0 seconds
```

However, this command only succeeds if the file is world readable. The following shows how to ensure the file is *not* world readable and illustrates dccp consequently failing to copy the file.

```
[user] $ chmod o-r my-test-file
[user] $ dccp dcap://adminNode/pnfs/site.de/data/my-test-file /tmp/test2.tmp
Command failed!
Server error message for [1]: "Permission denied" (errno 2).
Failed open file in the dCache.
Can't open source file : "Permission denied"
```

```
System error: Input/output error
```

This command did not succeed, because dCap access is unauthenticated and the user is mapped to a non-existent user in order to determine the access rights. However, you should be able to access the file with the NFS mount:

```
[user] $ dccp my-test-file /tmp/test2.tmp
541096 bytes in 0 seconds
```

If you have a valid grid proxy with a certificate subject which is properly mapped in the configuration file /opt/d-cache/etc/dcache.kpwd you can also try grid-authenticated access via the GSI-authenticated version of dCap:

```
[user] $ chgrp yourVO my-test-file
[user] $ export LD_LIBRARY_PATH=/opt/d-cache/dcap/lib/:$LD_LIBRARY_PATH
[user] $ dccp gsidcap://adminNode:22128/pnfs/site.de/data/my-test-file /tmp/test3.tmp
541096 bytes in 0 seconds
```

Or we let the SRM negotiate the protocol:

```
[user] $ export PATH=/opt/d-cache/srm/bin/:$PATH
[user] $ srmcp srm://adminNode:8443/pnfs/desy.de/data/my-test-file file:////tmp/test4.tmp
configuration file not found, configuring srmcp
created configuration file in ~/.srmconfig/config.xml
```

If the dCache instance is registered as a storage element in the LCG/EGEE grid and the LCG user interface software is available the file can be accessed via SRM:

```
[user] $ lcg-cp -v --vo yourVO \
srm://dCacheAdminFQN/pnfs/site.de/data/my-test-file \
file:///tmp/test5.tmp
Source URL: srm://dCacheAdminFQN/pnfs/site.de/data/my-test-file
File size: 541096
Source URL for copy: gsiftp://dCacheAdminFQN:2811//pnfs/site.de/data/my-test-file
Destination URL: file:///tmp/test5.tmp
# streams: 1
Transfer took 770 ms
```

and it can be deleted with the help of the SRM interface:

```
[user] $ srm-advisory-delete srm://dCacheAdminFQN:8443/pnfs/site.de/data/my-test-file
 srmcp error :  advisoryDelete(User [name=...],pnfs/site.de/data/my-test-file)
Error user User [name=...] has no permission to delete 000100000000000000BAF0C0
```

This works only if the grid certificate subject is mapped to a user which has permissions to delete the file:

```
[user] $ chown yourVO001 my-test-file
[user] $ srm-advisory-delete srm://dCacheAdminFQN:8443/pnfs/site.de/data/my-test-file
```

If the grid functionality is not required the file can be deleted with the NFS mount of the pnfs filesystem:

```
[user] $ rm my-test-file
```

The Web Interface for Monitoring dCache

In the standard configuration the dCache web interface is started on the head node and can be reached via port `2288`. Point a web browser to http://*`adminNode`*:2288/ [http://headNode:2288/] to get to the main menue of the dCache web interface. The contents of the web interface are self-explanatory and are the primary source for most monitoring and trouble-shooting tasks.

The “Cell Services” page displays the status of some important *cells* of the dCache instance. You might observe that some cells are marked “OFFLINE”. In general dCache has no knowledge about which cells are supposed to be online, but for purposes of monitoring, some cells may be hard coded in the file `/opt/d-cache/config/httpd.batch`:

```
#
create diskCacheV111.cells.WebCollectorV3 collector \
    "PnfsManager \
     PoolManager \
     -loginBroker=LoginBroker,srm-LoginBroker \
     -replyObject"
#
```

Additional cells may be added. To take effect, the `httpDomain` *domain* must be restarted by executing

```
[root] # /opt/d-cache/bin/dcache restart httpd
```

More information about the *`domainName`*`.batch` will follow in the next section.

The “Pool Usage” page gives a good overview of the current space usage of the whole dCache instance. In the graphs, free space is marked yellow, space occupied by *cached files* (which may be deleted when space is needed) is marked green, and space occupied by *precious files*, which cannot be deleted. Other states (e.g., files which are currently written) are marked purple.

The page “Pool Request Queues” (or “Pool Transfer Queues”) gives information about the number current requests handled by each pool. “Actions Log” keeps track of all the transfers performed by the pools up to now.

The remaining pages are only relevant with more advanced configurations: The page “Pools” (or “Pool Attraction Configuration”) can be used to analyze the current configuration of the *pool selection unit* in the pool manager. The remaining pages are relevant only if a *tertiary storage system (HSM)* is connected to the dCache instance.

Files

In this section we will have a look at the configuration and log files of dCache.

The dCache software is installed in one directory, normally `/opt/d-cache/`. All configuration files can be found here. In the following relative filenames will always be relative to this directory.

In the previous section we have already seen how a *domain* is restarted:

```
[root] # /opt/d-cache/bin/dcache restart domainName
```

Log files of domains are by default stored in `/var/log/`*`domainName`*`.log`. We strongly encourage to configure logrotate to rotate the dCache log files to avoid filling up the log file system. This can typically be achieved by creating the file `/etc/logrotate.d/dcache` with the following content:

```
/var/log/*Domain.log {
    compress
    rotate 100
    missingok
    copytruncate
}
```

The files `config/`*`domainName`*`Setup` contain configuration parameters for the domain. These files are typically symbolic links to `config/dCacheSetup`. This is the primary configuration file of dCache.

The only files which are different for each domain are `config/`*`domainName`*`.batch`. They describe which *cells* are started in the *domains*. Normally, changes in these files should not be necessary. However, if you need to change something, consider the following:

Since the standard `config/`*`domainName`*`.batch` files will be overwritten when updating to a newer version of dCache (e.g. with RPM), it is a good idea to modify only private copies of them. When choosing a name like `config/`*`newDomainName`*`.batch` you give the domain the name *`newDomainName`*. The necessary links can be created with

```
[root] # cd /opt/d-cache/config/
[root] # ../jobs/initPackage.sh
```

Then the old domain can be stopped and the new one started:

```
[root] # /opt/d-cache/bin/dache stop domainName
[root] # /opt/d-cache/bin/dcache start newDomainName
```

More details about domains and cells can be found in Chapter 5, *The Cell Package*.

The most central component of a dCache instance is the `PoolManager` cell. It reads additional configuration information from the file `config/PoolManager.conf` at start-up. However, in contrast to the `config/`*`domainName`*`Setup` files, it is not necessary to restart the domain when changing the file. We will see an example of this below.

Similar to `config/PoolManager.conf`, pools read their configuration from *`poolDir`*`/pool/setup` at startup.

The Admin Interface

Note

If you attempt to log into the admin interface without generating the `ssh`-keys you will get an error message.

```
[user] $ ssh -c blowfish -p 22223 -l admin headnode.example.org
Connection closed by 192.0.2.11
```

See ???.

dCache has a powerful administration interface. It is accessed with the `ssh` protocol. The server is part of the `adminDoor` domain. Connect to it with

```
[user] $ ssh -c blowfish -p 22223 -l admin headnode.example.org
```

The initial password is "`dickerelch`" (which is German for "fat elk") and you will be greeted by the prompt

```
   dCache Admin (VII) (user=admin)


(local) admin >
```

The password can now be changed with

```
(local) admin > cd acm
(acm) admin > create user admin
(acm) admin > set passwd -user=admin newPasswd newPasswd
(acm) admin > ..
(local) admin > logoff
```

This already illustrates how to navigate within the administration interface: Starting from the local prompt (`(local) admin >`) the command **cd** takes you to the specified *cell* (here `acm`, the access control manager). There two commands are executed. The escape sequence **..** takes you back to the local prompt and **logoff** exits the admin shell.

Note that **cd** only works from the local prompt. If the cell you are trying to access does not exist, the **cd** command will not complain. However, trying to execute any command subsequently will result in an error message "No Route to cell...". Type `..` to return to the `(local) admin >` prompt.

To create a new user, *`new-user`*, set a new password and to give him/her an access to a particular cell (for example to the `PoolManager`) run following command sequence:

```
(local) admin > cd acm
(acm) admin > create user new-user
(acm) admin > set passwd -user=new-user newPasswd newPasswd
(acm) admin > create acl cell.PoolManager.execute
(acm) admin > add access -allowed cell.PnfsManager.execute
```

Now you can check the permissions by:

```
(acm) admin > check cell.PnfsManager.execute new-user
Allowed
(acm) admin > show acl cell.PnfsManager.execute new-user
<noinheritance>
<new-user> -> true
```

Following commands allow to a particular user an access to every cell:

```
(acm) admin >  create acl cell.*.execute
(acm) admin > add access -allowed cell.*.execute new-user
```

To make an user as powerful as `admin` (dCache's equivalent to the `root` user):

```
(acm) admin > create acl *.*.*
(acm) admin > add access -allowed *.*.* new-user
```

All cells know the commands **info** for general information about the cell and **show pinboard** for listing the last lines of the *pinboard* of the cell. The output of these commands contains useful information for solving problems. It is a good idea to get aquainted with the normal output in the following cells: `PoolManager`, `PnfsManager`, and the pool cells (e.g., *`poolHostname`*`_1`).

If you want to find out which cells are running on a certain domain, you can issue the command **ps** in the `System` cell of the domain. For example, if you want to list the cells running on the `adminDoor`, **cd** to its `System` cell and issue the **ps** command.

```
(local) admin > cd System@adminDoorDomain
(System@adminDoorDomain) admin > ps
  Cell List
------------------
acm
alm
skm
c-dCacheDomain-101-102
System
c-dCacheDomain-101
RoutingMgr
alm-admin-103
pam
lm
```

The cells in the domain can be accessed using **cd** together with the cell-name scoped by the domain-name. So first, one has to get back to the local prompt, as the **cd** command will not work otherwise.

```
(System@adminDoorDomain) admin > ..
(local) admin > cd skm@adminDoorDomain
(skm) admin >
```

Note

If the cells are *well-known*, they can be accessed without adding the domain-scope. See Chapter 5, *The Cell Package* for more information.

The domains that are running on the dCache-instance, can be viewed in the layout-configuration (see Chapter 2, *Installing dCache*). Additionally, there is the `topo` cell, which keeps track of the instance's domain topology. If it is running, it can be used to obtain the list of domains the following way:

```
(local) admin > cd topo
(topo) admin > ls
dirDomain
infoDomain
adminDoorDomain
spacemanagerDomain
utilityDomain
gPlazmaDomain
nfsDomain
dCacheDomain
httpdDomain
statisticsDomain
```

```
namespaceDomain
```

Note

The `topo` cell rescans periodically which domains are running, so it can take some time until **ls** displays the full domain list.

There also is the command **help** for listing all commands the cell knows and their parameters. However, many of the commands are only used for debugging and development purposes. Only commands described in this documentation should be used for the administration of a dCache system.

The most useful command of the pool cells is **rep ls**. It lists the files which are stored in the pool by their `pnfs` IDs:

```
000100000000000000001120 <-P---------(0)[0]> 485212 si={myStore:STRING}
000100000000000000001230 <C----------(0)[0]> 1222287360 si={myStore:STRING}
```

Each file in a pool has one of the 4 primary states: “cached” (`<C---`), “precious” (`<-P--`), “from client” (`<--C-`), and “from store” (`<---S`).

Two commands in the pool manager are quite useful: **rc ls** lists the requests currently handled by the pool manager. A typical line of output for a read request with an error condition is (all in one line):

```
000100000000000000001230@0.0.0.0/0.0.0.0 m=1 r=1 [<unknown>]
  [Waiting 08.28 19:14:16]
  {149,No pool candidates available or configured for 'staging'}
```

As the error message at the end of the line indicates, no pool was found containing the file and no pool could be used for staging the file from a tertiary storage system.

Finally, **cm ls** with the option `-r` gives the information about the pools currently stored in the cost module of the pool manager. A typical output is:

```
(PoolManager) admin > cm ls -r
poolName1={R={a=0;m=2;q=0};S={a=0;m=2;q=0};M={a=0;m=100;q=0};PS={a=0;m=20;q=0};PC={a=0;m=20;q=0};
    (...continues...)    SP={t=2147483648;f=924711076;p=1222772572;r=0;lru=0;{g=20000000;b=0.5}}}
poolName1={Tag={{hostname=hostname}};size=0;SC=0.16221282938326134;CC=0.0;}
poolName2={R={a=0;m=2;q=0};S={a=0;m=2;q=0};M={a=0;m=100;q=0};PS={a=0;m=20;q=0};PC={a=0;m=20;q=0};
    (...continues...)    SP={t=2147483648;f=2147483648;p=0;r=0;lru=0;{g=4294967296;b=250.0}}}
poolName2microcebus_2={Tag={{hostname=hostname}};size=0;SC=2.7939677238464355E-4;CC=0.0;}
```

While the first line for each pool gives the information stored in the cache of the cost module, the second line gives the costs (SC: *space cost*, CC: *performance cost*) calculated for a (hypothetical) file of zero size. For details on how these are calculated and their meaning, see the section called “The Cost Module”.

The `ssh` admin interface can be used non-interactively by scripts. For this the dCache-internal `ssh` server uses public/private key pairs.

The file `config/authorized_keys` contains one line per user. The file has the same format as `~/.ssh/authorized_keys` which is used by **sshd**. The keys in `config/authorized_keys` have to be of type RSA1 as dCache only supports SSH protocol 1. Such a key is generated with

```
[user] $ ssh-keygen -t rsa1 -C 'SSH1 key of user'
Generating public/private rsa1 key pair.
Enter file in which to save the key (/home/user/.ssh/identity):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/identity.
Your public key has been saved in /home/user/.ssh/identity.pub.
The key fingerprint is:
c1:95:03:6a:66:21:3c:f3:ee:1b:8d:cb:46:f4:29:6a SSH1 key of user
```

The passphrase is used to encrypt the private key (now stored in `/home/user/.ssh/identity`). If you do not want to enter the passphrase every time the private key is used, you can use **ssh-add** to add it to a running **ssh-agent**. If no agent is running start it with

```
[user] $ if [ -S $SSH_AUTH_SOCK ] ; then echo "Already running" ; else eval `ssh-agent` ; fi
```

and add the key to it with

```
[user] $ ssh-add
Enter passphrase for SSH1 key of user:
Identity added: /home/user/.ssh/identity (SSH1 key of user)
```

Now, insert the public key `~/.ssh/identity.pub` as a separate line into `config/authorized_keys`. The comment field in this line "SSH1 key of *user*" has to be changed to the dCache user name. An example file is:

```
1024 35 141939124(... many more numbers ...)15331 admin
```

The key manager within dCache will read this file every minute.

Now, the ssh program should not ask for a password anymore. This is still quite secure, since the unencrypted private key is only held in the memory of the **ssh-agent**. It can be removed from it with

```
[user] $ ssh-add -d
Identity removed: /home/user/.ssh/identity (RSA1 key of user)
```

In scripts, one can use a "Here Document" to list the commands, or supply them to **ssh** as standard-input (stdin). The following demonstrates using a Here Document:

```
#!/bin/sh
#
#  Script to automate dCache administrative activity

outfile=/tmp/$(basename $0).$$.out

ssh -c blowfish -p 22223 admin@adminNode > $outfile << EOF
cd PoolManager
cm ls -r
(more commands here)
logoff
EOF
```

or, the equivalient as stdin.

```
#!/bin/bash
#
#   Script to automate dCache administrative activity.

echo -e 'cd pool_1\nrep ls\n(more commands here)\nlogoff' \
  | ssh -c blowfish -p 22223 admin@adminNode \
  | tr -d '\r' > rep_ls.out
```

The Graphical User Interface

Instead of using **ssh** to access the admin interface, the dCache graphical user interface can be used. If it is not included in the dCache distribution, it can be downloaded from the dCache homepage [http://www.dcache.org/]. It is started by

```
[user] $ java -jar org.pcells.jar
```

First, a new session has to be created with Session → New.... After giving the session a name of your choice, a login mask appears. The session is configured with the Setup button. The only thing that needs to be configured is the hostname. After clicking Apply and Quit you are ready to log in. Pressing the right mouse button clicking Login will scan the dCache instance for domains. Cells can be reached by clicking on their name and the same commands can be entered as in the SSH login.

The other tabs of the GUI are very useful for monitoring the dCache system.

Part II. Configuration of dCache

Table of Contents

This part contains descriptions of the components of dCache, their role, functionality within the framework. In short, all information necessary for configuring them.

Chapter 4. Configuration in pnfs

This chapter gives background information about pnfs, the filesystem, dCache is based on. Only the aspects of pnfs relevant to dCache will be explained here. A complete set of documentation is available from the pnfs homepage [http://www-pnfs.desy.de/].

The Use of pnfs in dCache

dCache uses pnfs as a filesystem and for storing meta-data. pnfs is a filesystem not designed for storage of actual files. Instead, pnfs manages the filesystem hierarchy and standard meta-data of a UNIX filesystem. In addition, other applications (as for example dCache) can use it to store their meta-data. pnfs keeps the complete information in a database.

pnfs implements an NFS server. All the meta-data can be accessed with a standard NFS client, like the one in the Linux kernel. After mounting, normal filesystem operations work fine. However, IO operations on the actual files in the pnfs will normally result in an error.

As a minimum, the pnfs filesystem needs to be mounted only by the server running the dCache core services. In fact, the pnfs server has to run on the same system. For details see (has to be written).

The pnfs filesystem may also be mounted by clients. This should be done by

```
[root] # mount -o intr,hard,rw pnfs-server:/pnfs /pnfs/site.de
```

(assuming the system is configured as described in the installation instructions). Users may then access the meta-data with regular filesystem operations, like **ls -l**, and by the pnfs-specific operations described in the following sections. The files themselves may then be accessed with the dCap protocol (see dCache Book Client Access and Protocols).

Mounting the pnfs filesystem is not necessary for client access to the dCache system if URLs are used to refer to files. In the grid context this is the preferred usage.

Communicating with the pnfs Server

Many configuration parameters of pnfs and the application-specific meta-data is accessed by reading, writing, or creating files of the form .(*command*)(*para*). For example, the following prints the pnfsID of the file /pnfs/site.de/some/dir/file.dat:

```
[user] $ cat /pnfs/site.de/any/sub/directory/'.(id)(file.dat)'
000400000000000002320B8
[user] $
```

From the point of view of the NFS protocol, the file .(id)(file.dat) in the directory /pnfs/site.de/some/dir/ is read. However, pnfs interprets it as the command id with the parameter file.dat executed in the directory /pnfs/site.de/some/dir/. The quotes are important, because the shell would otherwise try to interpret the parentheses.

Some of these command-files have a second parameter in a third pair of parentheses. Note, that files of the form `.(command)(para)` are not really files. They are not shown when listing directories with **ls**. However, the command-files are listed when they appear in the argument list of **ls** as in

```
[user] $ ls -l '.(tag)(sGroup)'
-rw-r--r--   11 root     root           7 Aug  6  2004 .(tag)(sGroup)
```

Only a subset of file operations are allowed on these special command-files. Any other operation will result in an appropriate error. Beware, that files with names of this form might accidentally be created by typos. They will then be shown when listing the directory.

pnfsIDs

Each file in pnfs has a unique 12 byte long pnfsID. This is comparable to the inode number in other filesystems. The pnfsID used for a file will never be reused, even if the file is deleted. dCache uses the pnfsID for all internal references to a file.

The pnfsID of the file *filename* can be obtained by reading the command-file `.(id)(`*filename*`)` in the directory of the file.

A file in pnfs can be referred to by pnfsID for most operations. For example, the name of a file can be obtained from the pnfsID with the command nameof as follows:

```
[user] $ cd /pnfs/site.de/any/sub/directory/
[user] $ cat '.(nameof)(000400000000000002320B8)'
file.dat
```

And the pnfsID of the directory it resides in is obtained by:

```
[user] $ cat '.(parent)(000400000000000002320B8)'
000400000000000001DC9E8
```

This way, the complete path of a file may be obtained starting from the pnfsID. Precisely this is done by the tool **pathfinder**:

```
[user] $ . /usr/etc/pnfsSetup
[user] $ PATH=$PATH:$pnfs/tools
[user] $ cd /pnfs/site.de/another/dir/
[user] $ pathfinder 000400000000000002320B8
000400000000000002320B8 file.dat
000400000000000001DC9E8 directory
000400000000000000001060 sub
000100000000000000001060 any
000000000000000000001080 usr
000000000000000000001040 fs
000000000000000000001020 root
000000000000000000001000 -
000000000000000000000100 -
000000000000000000000000 -
/root/fs/usr/any/sub/directory/file.dat
```

The first two lines configure the pnfs-tools correctly. The path obtained by **pathfinder** does not agree with the local path, since the latter depends on the mountpoint (in the example /pnfs/site.de/). The pnfsID corresponding to the mountpoint may be obtained with

```
[user] $ cat '.(get)(cursor)'
dirID=000400000000000001DC9E8
dirPerm=0000001400000020
mountID=00000000000000000001080
```

The dirID is the pnfsID of the current directory and mountID that of the mountpoint. In the example, the pnfs server path /root/fs/usr/ is mounted on /pnfs/site.de/.

Directory Tags

In the pnfs filesystem, each directory has a number of tags. The existing tags may be listed with

```
[user] $ cat '.(tags)()'
.(tag)(OSMTemplate)
.(tag)(sGroup)
```

and the content of a tag can be read with

```
[user] $ cat '.(tag)(OSMTemplate)'
StoreName myStore
```

A nice trick to list all tags with their contents is

```
[user] $ grep "" $(cat ".(tags)()")
.(tag)(OSMTemplate):StoreName myStore
.(tag)(sGroup):STRING
```

Directory tags may be used within dCache to control which pools are used for storing the files in the directory (see the section called “The Pool Selection Mechanism”). They might also be used by a *tertiary storage system* for similar purposes (e.g. controlling the set of tapes used for the files in the directory).

Even if the directory tags are not used to control the bahaviour of dCache, some tags have to be set for the directories where dCache files are stored. The installation procedure takes care of this: In the directory /pnfs/*site.de*/data/ two tags are set to default values:

```
[user] $ cd /pnfs/site.de/data/
[user] $ grep "" $(cat ".(tags)()")
.(tag)(OSMTemplate):StoreName myStore
.(tag)(sGroup):STRING
```

The following directory tags appear in the dCache context:

Directory Tags for dCache

OSMTemplate — Contains one line of the form “StoreName *storeName*” and specifies the name of the store that is used by dCache to construct the *storage class* if the *HSM type* is osm.

hsmType — The *HSM type* is normally determined from the other existing tags. E.g., if the tag OSMTemplate exists, HSM type osm is assumed. With this tag it can be set explicitly. An class implementing that HSM type has to exist. Currently the only implementations are osm and enstore.

sGroup — The storage group is also used to construct the *storage Class* if the *HSM type* is osm.

cacheClass — The cache class is only used to control on which pools the files in a directory may be stored, while the storage class (constructed from the two above tags) might also be used by the HSM. The cache class is only needed if the above two tags are already fixed by HSM usage and more flexibility is needed.

hsmInstance — If not set, the HSM instance will be the same as the HSM type. Setting this tag will only change the name as used in the *storage class* and in the pool commands.

There are more tags used by dCache if the HSM type `enstore` is used.

When creating or changing directory tags by writing to the command-file as in

```
[user] $ echo 'content' > '.(tag)(tagName)'
```

one has to take care not to treat the command-files in the same way as regular files, because tags are different from files in the following aspects:

1. The *tagName* is limited to 62 characters and the *content* to 512 bytes. Writing more to the command-file, will be silently ignored.

2. If a tag which does not exist in a directory is created by writing to it, it is called a *primary* tag.

   Removing a primary tag invalidates this tag. An invalidated tag behaves as if it does not exist. All filesystem IO operations on that tag produce an “File not found” error. However, a lookup operation (e.g. ls) will show this tag with a 0 byte size. An invalidated tag can be revalidated with the help of the tool **repairTag.sh** in the `tools/` directory of the `pnfs` distribution. It has to be called in the directory where the primary tag was with the tag name as argument.

3. Tags are *inherited* from the parent directory by a newly created directory. Changing a primary tag in one directory will change the tags inherited from it in the same way, even if it is invalidated or revalidated. Creating a new primary tag in a directory will not create a inherited tag in its subdirectories.

   Moving a directory within the `pnfs` filesystem will not change the inheritance. Therefore, a directory does not necessarily inherit tags from its parent directory. Removing an inherited tag does not have any effect.

4. Writing to an inherited tag in the subdirectory will break the inheritance-link. A *pseudo-primary* tag will be created. The directories which inherited the old (inherited) tag will inherit the pseudo-primary tag. A pseudo-primary tag behaves exactly like a primary tag, except that the original inherited tag will be restored if the pseude-primary tag is removed.

If directory tags are used to control the behaviour of dCache and/or a tertiary storage system, it is a good idea to plan the directory structure in advance, thereby considering the necessary tags and how they should be set up. Moving directories should be done with great care or even not at all. Inherited tags can only be created by creating a new directory.

Global Configuration with *Wormholes*

`pnfs` provides a way to distribute configuration information to all directories in the `pnfs` filesystem. It can be accessed in a subdirectory `.(config)()` of any `pnfs`-directory. It behaves similar to a hardlink.

In the default configuration this link points to /pnfs/fs/admin/etc/config/. In it are three files: '.(config)()'/serverId contains the domain name of the site, '.(config)()'/serverName the fully qualified name of the pnfs server, and '.(config)()'/serverRoot should contain "000000000000000000001080 .".

The dCache specific configuration can be found in '.(config)()'/dCache/dcache.conf. This file contains one line of the format *hostname*:*port* per dCap door which may be used by dCap clients when not using URLs. The **dccp** program will choose randomly between the doors listed here.

Normally, reading from files in pnfs is disabled. Therefore it is necessary to switch on I/O access to the files in '.(config)()'/ by e.g.:

```
[root] # touch '.(config)()/.(fset)(serverRoot)(io)(on)'
```

After that, you will notice that the file is empty. Therefore, take care, to rewrite the information.

Deleted Files in pnfs

When a file in the pnfs filesystem is deleted the server stores information about is in the subdirectories of /opt/pnfsdb/pnfs/trash/. For dCache, the cleaner cell in the pnfsDomain is responsible for deleting the actual files from the pools asyncronously. It uses the files in the directory /opt/pnfsdb/pnfs/trash/2/. It contains a file with the pnfs ID of the deleted file as name. If a pool containing that file is down at the time the cleaner tries to remove it, it will retry for a while. After that, the file /opt/pnfsdb/pnfs/trash/2/current/failed.*poolName* will contain the pnfs IDs which have not been removed from that pool. The cleaner will still retry the removal with a lower frequency.

Access Control

The files /pnfs/fs/admin/etc/exports/*hostIP* and /pnfs/fs/admin/etc/exports/*netMask*.*netPart* are used to control the host-based access to the pnfs filesystem via mount points. They have to contain one line per NFS mount point. The lines are made of the following four space-separated fields fields:

- Mount point for NFS (the part after the colon in e.g. host:/mountpoint)
- The virtual PNFS path which is mounted
- Permission: 0 means all permissions and 30 means disabled I/O.
- Options (should always be nooptions)

In the initial configuration there is one file /pnfs/fs/admin/etc/exports/0.0.0.0..0.0.0.0 containing

```
/pnfs /0/root/fs/usr/ 30 nooptions
```

thereby allowing all hosts to mount the part of the pnfs filesystem containing the user data. There also is a file /pnfs/fs/admin/etc/exports/127.0.0.1 containing

```
/fs /0/root/fs 0 nooptions
/admin /0/root/fs/admin 0 nooptions
```

The first line is the mountpoint used by the admin node. If the pnfs mount is not needed for client operations (e.g. in the grid context) and if no *tertiary storage system (HSM)* is connected, the file /pnfs/fs/admin/etc/exports/0.0.0.0..0.0.0.0 may be deleted. With an HSM, the pools which write files into the HSM have to mount the pnfs filesystem and suitable export files have to be created.

In general, the user ID 0 of the root user on a client mounting the pnfs filesystem will be mapped to nobody (not to the user nobody). For the hosts whose IP addresses are the file names in the directory /pnfs/fs/admin/etc/exports/trusted/ this is not the case. The files have to contain only the number 15.

The Databases of pnfs

pnfs stores all the information in GNU dbm database files. Since each operation will lock the database file used globally and since GNU dbm cannot handle database files larger than 2GB, it is advisable to "split" them sutably to future usage. Each database stores the information of a sub-tree of the pnfs filesystem namespace. Which database is responsible for a directory and subsequent subdirectories is determined at creation time of the directory. The following procedure will create a new database and connect a new subdirectory to it.

Each database is handled by a separate server process. The maximum number of servers is set by the variable shmservers in file /usr/etc/pnfsSetup. Therefore, take care that this number is always higher than the number of databases that will be used (restart pnfs services, if changed).

Prepare the environment with

```
[root] # . /usr/etc/pnfsSetup
[root] # PATH=${pnfs}/tools:$PATH
```

To get a list of currently existing databases, issue

```
[root] # mdb show
ID Name Type Status Path
------------------------------
0 admin r enabled (r) /opt/pnfsdb/pnfs/databases/admin
1 data1 r enabled (r) /opt/pnfsdb/pnfs/databases/data1
```

Choose a new database name *databaseName* and a location for the database file *databaseFilePath* (just a placeholder for the PostgreSQL version of pnfs) and create it with

```
[root] # mdb create databaseName databaseFilePath
```

e.g.

```
[root] # mdb create data2 /opt/pnfsdb/pnfs/databases/data2
```

Make sure the file *databaseFilePath* exists with

```
[root] # touch databaseFilePath
```

This might seem a little strange. The reason is that the PostgreSQL version of the pnfs server only uses the file as reference and stores the actual data in the PostgreSQL server.

In order to refresh database information run

```
[root] # mdb update
Starting data2
```

Running command **mdb show** shows the new database:

```
[root] # mdb show
ID Name Type Status Path
------------------------------
0 admin r enabled (r) /opt/pnfsdb/pnfs/databases/admin
1 data1 r enabled (r) /opt/pnfsdb/pnfs/databases/data1
2 data2 r enabled (r) /opt/pnfsdb/pnfs/databases/data2
```

In the pnfs filesystem tree, create the new directory in the following way

```
[root] # cd /pnfs/site.de/some/sub/dir/
[root] # mkdir '.(newDbID)(newDirectory)'
```

where *newDbID* is the ID of the new database as listed in the output of **mdb show** and *newDirectory* is the name of the new directory. E.g.

```
[root] # cd /pnfs/desy.de/data/zeus/
[root] # mkdir '.(2)(mcdata)'
```

The new database does not know anything about the wormhole '.(config)()', yet. For this, the pnfs ID of the wormhole directory (/pnfs/fs/admin/etc/config/) has to be specified. It can be found out with

```
[root] # sclient getroot ${shmkey} 0
0 000000000000000000001000 wormholePnfsId
```

The last pnfsID is the one of the wormhole directory of the database with ID 0 (already set correctly). Now you can set this ID with

```
[root] # sclient getroot ${shmkey} newDbID wormholePnfsId
newDbID 000000000000000000001000 wormholePnfsId
```

For example, do the following

```
[root] # sclient getroot ${shmkey} 0
0 000000000000000000001000 0000000000000000000010E0
[root] # sclient getroot ${shmkey} 2 0000000000000000000010E0
2 000000000000000000001000 0000000000000000000010E0
```

Finally, add directory tags for the new directories. The default tags are added by

```
[root] # cd /pnfs/site.de/some/sub/dir/newDirectory
[root] # echo 'StoreName myStore' > '.(tag)(OSMTemplate)'
[root] # echo 'STRING' > '.(tag)(sGroup)'
```

Chapter 5. The Cell Package

Apart from the `pnfs` server, all of dCache makes use of the cell package. It is a framework for a distributed and scalable server system in Java. The dCache system divided into cells which communicate with each other via messages. Several cells run simultaneously in one domain.

Each domain runs in a separate Java virtual machine and each cell is run as a separate thread therein. The domains communicate with each other via TCP connections which are established at start-up. In the standard setup all domains connect with the `dCacheDomain` which routes all messages to the appropriate domains. Note, that actual data transfers are not done via these established connections.

At start-up a domain asks the `serviceLocatorHost` on the `serviceLocatorPort` (as configured in `config/`*`domainName`*`Setup`) for a domain to connect to. This request is handled by the location manager in the `lmDomain` In the standard setup it will tell all other domains to connect to the `dCacheDomain` and will provide the necessary connection information. A TCP connection between the new domain and the `dCacheDomain` will be established.

Within this framework, cells send messages to other cells addressing them in the form *`cellName`*`@`*`domainName`*. This way, cells can communicate without knowledge about the host they run on. Some cells are *well known*, i.e. they can be addressed just by their name without `@`*`domainName`*. Evidently, this can only work properly if the name of the cell is unique throuout the hole system.

If two well known cells with the same name are present, the system will behave in an undefined way. Therefore it is wise to take care when starting, naming, and renaming the well known cells. Some cell types, like the `PoolManager` will always be well known, while others may be made well known by the `-export` option. Some cell types can and should never be well known. The same problem can arise if two dCache instances which are meant to be separate use the same *location manager* due to a miss-configuration: Messages meant for e.g. the pool manager of one instance get routed to the pool manager of the other instance which generally will not be able to handle the request properly.

A domain is started with a shell script `jobs/`*`domainName`* (which in the standard setup is for all domains a link to `jobs/wrapper2.sh`). This will use the configuration variables in `config/`*`domainName`*`-Setup` (which normally is a link to `config/dCacheSetup`), start the Java virtual machine and execute the cell package commands in `config/`*`domainName`*`.batch`. The main task of these commands is to start up all the cells which should be running in the domain. It follows a typical batch file.

Example 5.1. Example batch file `config/gridftpdoor.batch`

```
set printout default 2
set printout CellGlue none
onerror shutdown

check -strong setupFile
copy file:${setupFile} context:setupContext
import context -c setupContext
check -strong serviceLocatorPort serviceLocatorHost
check -strong sshPort ftpPort

create dmg.cells.services.RoutingManager  RoutingMgr
create dmg.cells.services.LocationManager lm \
       "${serviceLocatorHost} ${serviceLocatorPort}"

create dmg.cells.services.login.LoginManager GFTP \
            "2811 \
             -export \
             diskCacheV111.doors.GsiFtpDoorV1 \
             -prot=raw \
             -clientDataPortRange=${clientDataPortRange} \
             -root=${ftpBase} \
             -kpwd-file=${kpwdFile} \
             -tlog=/tmp/dcache-ftp-tlog \
             -maxLogin=100 \
             -brokerUpdateTime=5 \
             -protocolFamily=gsiftp \
             -loginBroker=LoginBroker \
             -poolManagerTimeout=5400 \
             -pnfsTimeout=120 \
             -maxRetries=80 \
             -maxStreamsPerClient=10 \
             -ftp-adapter-internal-interface=10.0.1.1 \
"
```

It is not necessary to understand every detail of the syntax of the batch files even for the most advanced dCache administration tasks. The following explanations should be sufficient: The first tree lines set some logging behaviour.

The next 5 lines import the variables from the `config/`*`domainName`*`Setup` file into the *context* of the domain. The values of the variables defined in the setup file are placed whereever `${`*`variableName`*`}` appears.

The **create** commands at the end start three cells. It takes up to three parameters: The Java class to start, the name of the cell and the argument string which is passed to the class. In the standard setup, most parameters to the cells are either set to reasonable values or are filled from variables defined in the setup file. This way, the batch files only need to be changed for more advanced configuration tasks.

The routing manager and location manager cells are started in each domain are part of the underlying cell package structure. Each domain will contain at least one cell in addition to them.

Chapter 6. Resilience with the Replica Manager

Alex Kulyavtsev

Mathias de Riese

If no *tertiary storage system* is connected to a dCache instance (i.e., it is configured as a *large file store*), there might be only one copy of each file on disk. (At least the *precious replica*.) If a higher security and/or availability is required, the resilience feature of dCache can be used: If running in the default configuration, the replica manager will make sure that the number of *replicas* of a file will be at least 2 and not more than 3. If only one replica is present it will be copied to another pool by a *pool to pool transfer*. If 4 or more replicas exist, some of them will be deleted.

Installation

To activate Replica Manager you need make changes in all 3 places:

1. `etc/node_config` on *all admin nodes* in this dCache instance.
2. `replicaSetup` file on node where replica manager is runnig
3. define Resilient pool group(s) in `PoolManager.conf`

```
#  - - - - Will Replica Manager be started?
#   Values:  no, yes
#   Default: no
#
```

This has to be set to “yes” on every node, if there is a replica manager in this dCache instance. Where the replica manager is started is controlled in `etc/node_config`. If it is not started and this is set to “yes” there will be error messages in `log/dCacheDomain.log`. If this is set to “no” and a replica manager is started somewhere, it will not work properly.

```
#replicaManager=no

#  - - - - Which pool-group will be the group of resilient pools?
#   Values:  <pool-Group-Name>, a pool-group name existing in the PoolManager.conf
#   Default: ResilientPools
#
```

Only pools defined in pool group ResilientPools in `config/PoolManager.conf` will be managed by ReplicaManager. You shall edit `config/PoolManager.conf` to make replica manager work. To use another pool group defined in `PoolManager.conf` for replication, please specify group name by changing setting :

```

#resilientGroupName=ResilientPools
```

Please scroll down “replica manager tuning” make this and other changes.

Starting the Replica Manager

Beginning with version 1.6.6 of dCache the replica manager can be started as follows:

The replica manager will use the same PostgreSQL database and database user srmdcache as the SRM. The standard configuration assumes that the database server is installed on the same machine as the replica manager — usually the admin node of the dCache instance. To create and configure the database *replicas* used by the replica manager in the database server do:

```
[root] # su postgres
[user] $ createdb -U srmdcache replicas
[user] $ psql -U srmdcache -d replicas -f /opt/d-cache/etc/psql_install_replicas.sql
[user] $ exit
```

The start-up script `bin/dcache-core` already contains the correct lines to start and stop the domain containing the replica manager as comments. Just remove the two hash (“#”) signs and restart the dCache instance. The replica manager may also be started separately by

```
[root] # /opt/d-cache/jobs/replica -logfile=/opt/d-cache/log/replica.log start
```

and stopped by

```
[root] # /opt/d-cache/jobs/replica stop
```

In the default configuration, all pools of the dCache instance will be managed. The replica manager will keep the number of replicas between 2 and 3 (including). At each restart of the replica manager the pool configuration in the database will be recreated.

Operation

When file is transfered into the dCache its replica is copied into one of the pools. Since this is the only replica and normally required range is higher (e.g., (2,3)), this file will be replicated to other pools. When some pool goes down the replica count for the files in that pool may fall below the valid range and these files will be replicated. Replicas of the file with replica count below the valid range and which need replication are called *deficient replicas*.

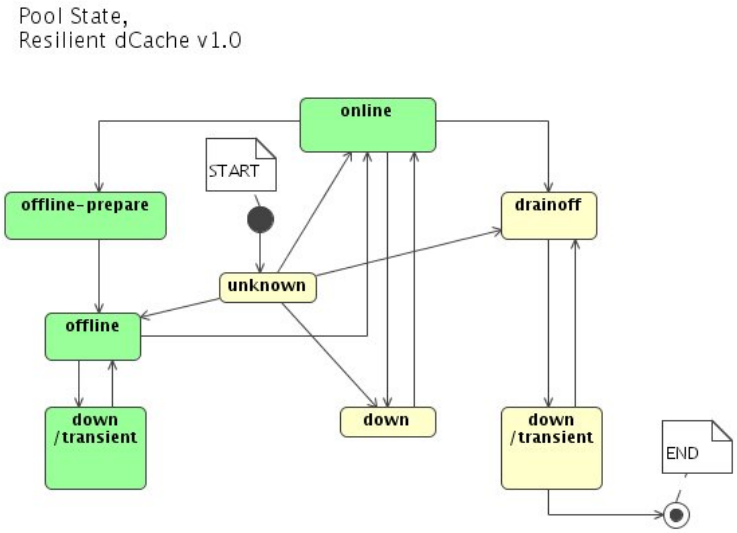
Later on some of the failed pools can come up and bring online more valid replicas. If there are too many replicas for some file these extra replicas are called *redundant replicas* and they will be “reduced”. Extra replicas will be deleted from pools.

Resilience Manager (RM) counts number of replicas for each file in the pools which can be used online (see Pool States below) and keeps number of replicas within the valid range (min, max).

RM keeps information about pool state, list of the replicas (file ID, pool) and current copy/delete operations in persistent database.

For each replica RM keeps list of pools where it can be found. For the pools pool state is kept in DB. There is table which keeps ongoing operations (replication, deletion) for replica.

Figure 6.1. Pool State Diagram



This is description of pool states as it is in v1.0 of Risilience Manager. Some of the states and transitions will be changed in the next release.

online — normal operation. Replicas in this state are readable and can be counted. Files can be written (copied) to this pool.

down — dCache pool is stopped by operator or crashed. On startup, pool comes briefly to the online state, and then it goes “down” to do pool “Inventory” — to cleanup files which broke when pool crashed during transfer. When pool comes online again, RM will update list of replicas in the pool and store it in the DB.

Replicas in pools which are “down” are not “counted”, so when pool crashes it reduces number of “online” replicas for some files. The crash of the pool (pool departure) may trigger replication of multiple files.

Pool recovery (arrival) may rigger massive deletition of file replicas, not necessarily in this pool.

There are special situations when operator wants to change pool state and he/she does not want to trigger massive replication. Or vice versa he/she wants to take pool permanently out of operation and wants to make sure that files in the pool will not be locked out and will be available later.

offline — replicas in this pool are counted whether this pool is up or down. It does done matter fore replication purpose if offline pool goes down or up. Rationale — operator wants to bring pool down briefly and he/she knows that replicas in the pool are safe. This state is introduced to avoid unnecessary massive replication. When pool comes online from offline state replicas in the pool will be inventoried to make sure we know the real list of replicas in the pool.

down	operator needs to set pool or set of pools down permanently and wants to make sure that there no replicas “locked out” when all known replicas of the file are in the pools which are unavailable. Thus whether pool is really up or down replicas in it are not counted.
drainoff, offline-prepare	transient states between online and down or offline states respectively. If there are files which can be “locked out” in down or offline states, they will be evacuated — at least one replica for each locked file will be copied out. It is unlikely that file will be locked out when singly pool goes down — normally few replicas are online. When several pools go down or set drainoff or offline file lockout may happens.

Note

Currently replicas counted separately in groups of offline-prepare and drainoff pools.

RM needs the single copy of the replica to be copied out and then you can turn pool down, the other replicas will be made from the replica available online. To confirm that it is safe to turn pool down there is command to check number of files which can be locked in this pool.

v1.0 — these states called “transient” but pool does not automatically turned down

Startup

The number of the pools in the system may be large and it may be inconvenient to keep configuration of the system predefined in some file. On startup complete configuration is unknown and RM tries to keep number of replicas in the valid range as pools arrive and departure and files are copied in. On the other hand when groups of pools arrive or departure it leads to massive replica cloning or reduction. It is beneficial to suspend ajustments until system arrives to more or less stable configuration.

When RM starts it cleans up DB. Then it waits for some time to give a chance to the pools to get connected. RM tries do not start too early and give a chance to most of the pools in the system to connect. Otherwise unnecessary massive replication will start. When configuration is unknown RM waits for some time until “quorum” of the pools get connected. Currently this is implemented by some delay to start adjustments to get chance to the pools to connect.

Normally (during Cold Start) all information in DB is cleaned up and recreated again by polling pools which are online shortly after some minimum delay after RM starts. RM starts to track pools state (pool up/down messages and polling list of online pools) and updates list of replicas in the pools which came online. This process lasts for about 10-15 minutes to make sure all pools come up online and/or get connected. Pools which once get connected to RM are in online or down state.

It can be annoying to wait for some large period of time until all known “good” pools get connected. There is “Hot Restart” option to accelerate restart of the system after the crash of the head node.

On Hot Restart RM retrieves information about pools state before the crash from DB and saves pools state to some internal structure. When pool gets connected RM checks the old pool state and registers old pools state in DB again if the state was offline, offline-prepare or “drainoff” state. RM also checks if the pool was online before the crash. When all pools which were “online” get connected once, RM supposes it recovered it’s old configuration and RM starts adjustments. RM operates in the “fluid world”. It does not required that

pools stay online. The point is when all online pools get connected online we can start adjustments. If some pools went down during connection process they are already accounted and adjustment will take care of it.

Example: Suppose we had have ten pools in the system where eight pools were online and two were offline. RM does not care about two offline pools get connected to start adjustments. For the other eight pools which were online, suppose one pool get connected and then it falls down while the other pools try to connect. RM considers this pool in known state, and when other seven pools get connected it can start adjustments and does not wait any more. If system was in equilibrium state before the crash, RM may find some deficient replicas because of the crashed pool and start replication right away.

More on operation

RM has few threads running at the same time. Adjuster keeps count of the replicas within the valid range, the other threads help to do this.

Adjuster. Information about all replicas is kept in DB. Adjuster makes several queries in DB during adjustment cycle to get the list of files for which replicas must be reduced or replicated:

- redundant replicas, Nrep > max
- unique replicas in drainoff pools
- unique replicas in offline-prepare pools
- deficient replicas, Nrep < min

Number of replicas is counted in pools which are online or offline. Offline-prepare or drainoff pools considered read-only and can be used as a source pool for replication. Last replica of the file in the system must not be removed.

The information in DB updated when new replica is added or removed from the pool. When some pool changes it's state all replicas in the pool became available or unavailable. This changes the number of accessible replicas for the file. The current list is marked as invalid and RM restarts adjustment cycle from the beginning. When nothing happens for some time adjustment cycle is triggered by timeout to make sure RM did not miss anything because some messages get lost.

When it is found that replica needs replication or reduction the worker thread starts to do the job asynchronously. Number of Worker threads is limited to the max [default=6], separately for reducers and replicators. If no workers are available adjuster will wait for the worker thread. Worker thread starts operation by sending message to dCache and waits until operation finishes or timeout expires. The timeout is different for reduction (replica removal) and replication, the replication timout shall be larger to account for the time to transfer the largest file between the pools. When the worker thread starts operation it marks replica as "having the operation" in action table in DB, and this replica will be excluded from other operations in the system until operation done or timeout expire. When there are few replicas for the same file found to be replicated (or reduced), RM schedules one replica for replication and proceeds with processing the other files. When Adjuster reaches the end of the list, it may return to the processing of the other replicas of the first file without delay considering the previous operation with the file complete.

Sometimes Adjuster gets error on operation with replica and in some cases if it does the same operation with the same replica again this "unresolved" error happens again and again blocking RM to keep from processing

other replicas. To avoid such loops and “dynamic deadlock” RM can put the replica which encountered the problem into *“exclude”* state. To return this replica into operation administrator shall manually *“release”* this replica.

When pool changes its state RM receives a message which can be lost or is not sent in some cases like pool crash. To make sure RM has correct information about pool states it runs PoolsWatchDog thread. WatchDog polls pools states and compares it to the result of the previous poll to find out which pools departed from or arrived into the system. Then it sleeps for some time and does the check again. When there were no changes in the pool configuration WatchDog throttles messages “no pool configuration change” in the log file — but it is still running.

Cyclical threads — Adjuster and WatchDog write and timestamps it’s current state in DB. It is displayed on Web page so it is possible to check if it is running. Excluded files are listed there too.

Commands

If you are advanced user and have proper privileges and you know how to issue command to admin interface you may connect to the `ReplicaManager` cell and issue the following commands. You may find more commands in online help which are for debug only — do not use them as they can stop RM operating properly.

`set pool` ***`pool state`***	set pool state
`show pool` ***`pool`***	show pool state
`ls unique` ***`pool`***	check if pool drained off (has unique pndfsIds). Reports number of replicas in this pool. Zero if no locked replicas.
`exclude` ***`pnfsId`***	exclude ***`pnfsId`*** from adjustments
`release` ***`pnfsId`***	removes transaction/’BAD’ status for pnfsId
`debug true | false`	enable/disable DEBUG messages in the log file

Hybrid dCache

“Hybrid” dCache operates on combination of “normal” pools (backuped to the tape or “scratch” pools) and the set of resilient pools. Resilience manager takes care only for the subset of pools configured in the Pool Group named “ResilientPools” and ignores all other pools. Currently resilient pool group name is hardcoded as “ResilientPools”, and you shall create replica manager cell to use in hybrid dCache by instantiating class diskCacheV111.replicaManager.ReplicaManagerV2 (note “V2” in the classname).

Add to `PoolManager.conf`:

```
psu create pgroup ResilientPools
```

```
psu addto  pgroup ResilientPools myPoolName001
psu addto  pgroup ResilientPools myPoolName002
psu addto  pgroup ResilientPools myPoolName003
```

Pools included in the resilient pool groop can also be included in other pool groups.

Arguments for the `ReplicaManager` cell in the batch file:

Default argument values as for `$Id: ReplicaManager.java,v 1.22 2004/08/25 22:32:07 cvs Exp $`

You do not need to put these arguments in the batch file until you want to change these defaults and you know what are you doing. For normal operation you may want to chose “-ColdStart” or “-hotRestart” (is default) mode of startup and (min,max) for desired range of number of replicas of the file.

General	
-min=2 -max=3	Valid range for the replicas count in “available” pools.
-debug=false | true	Disable / enable debug messages in the log file
Startup mode	
-hotRestart	
default	Startup will be accelerated, when all “known” pools registered in DB as “online” before the crash, will re-connect again during hot restart. Opposite to -coldStart.
-coldStart	
optional	Good for the first time or big changes in pool configuration. Will create new pool configuration in DB. Opposite to -hotRestart.
-delayDBStartTO=1200	on Cold Start:
20 min	DB init thread sleep this time to get chance to pools to get connected to prevent massive replications when not all pools connected yet when the replication starts.
-delayAdjStartTO=1260	Normally Adjuster waits for DB init thread to finish. If by some abnormal reason it can not find DB thread then it will sleep for this delay. It should be slightly more then “delayDBStartTO”.
21 min	
DB connection	
-dbURL=jdbc:postgresql:// dbservernode.domain.edu:5432/replicas	Configure host:port where DB server is running and DB table name. For DB on remote host you shall enable `TCP` connections to DB from your host (see installation instructions).
-jdbcDrv=org.postgresql.Driver	DB driver. Replica Manager was tested with Postgres DB only.
-dbUser=myDBUserName	Configure different DB user
-dbPass=myDBUserPassword	Configure different DB path
Delays	
-maxWorkers=4	Number of worker threads to do the replication, the same number of worker threads used for reduction. Must be more for larger system but avoid situation when requests get queued in the pool.
-waitReplicateTO=43200	

12 hours	Timeout for pool-to-pool replica copy transfer.
-waitReduceTO=43200	
12 hours	Timeout to delete replica from the pool.
-waitDBUpdateTO=600	
10 min	Adjuster cycle period. If nothing changed, sleep for this time, and restart adjustment cycle to query DB and check do we have work to do?
-poolWatchDogPeriod=600	
10 min	

Pools Watch Dog pool period. Poll the pools with this period to find if some pool went south without sending notice (messages). Can not be too short because pool can have high load and do not send pings for some time. Can not be less than pool ping period.

Monitoring Installation

DRAFT

Scope of this document

This section briefly summarizes steps to install Monitoring for the Replica Manager. RM installation is described here the section called “Installation”. It’s meant as “aide-memoire” for people doing dCache packaging. The document is of very little use for dCache end users. You may find useful information on how to operate the Resilience Manager at the Resilient Manual.

Resilience Manager uses Tomcat to monitor its operation. This package is not required for normal RM operation, but it is highly desirable to install and run it to properly monitor RM.

Prerequisites

The PostgreSQL database must be installed and running on the machine hosting the replica manager module and DB schema must be initalized as described in RM installation instructions (“Database Preparation”). You will see something in the tables if Resilience Manager is running.

Tomcat Installation and Configuration

- get the binary Tomcat distribution (currently version 5.5.x) from Apache Jakarta Tomcat website http://tomcat.apache.org/download-55.cgi#5.5.25 and install it folowing the instruction form the web site.

- Tomcat uses port 8080 by default, but we have changed it to 8090 in the `conf/server.xml` file because 8080 is too popular — check this port and change it !

```
<Connector port="8090"
...
```

- You need to copy jdbc PostgreSQL driver into `common/lib` directory in Tomcat installation from URL : http://jdbc.postgresql.org/download/pg74.215.jdbc3.jar This version of the driver works with Java 1.4 and 1.5 .
- deploy `replica.war` file into `tomcat/apache-tomcat-5.5.x/webapps/`
- start the Tomcat:

```
[root] # tomcat/apache-tomcat-5.5.x/bin/startup.sh
```

You can now access the Resilience Manager monitoring info using URL: http://*your.hostname:8090*/replica

Chapter 7. Configuring the Pool Manager

The heart of a dCache system is the *pool manager*. When a user performs an action on a file - reading or writing - a *transfer request* is sent to the dCache system. The pool manager then decides how to handle this request.

If a file the user wishes to read resides on one of the storage-pools within the dCache system, it will be transferred from that pool to the user. If it resides on several pools, the file will be retrieved from the pool which is least busy. If all pools the file is stored on are busy, a new copy of the file on an idle pool will be created and this pool will answer the request.

A new copy can either be created by a *pool to pool transfer* (p2p) or by fetching it from a connected *tertiary storage system* (sometimes called HSM - hierarchical storage manager). Fetching a file from a tertiary storage system is called *staging*. It is also performed if the file is not present on any of the pools in the dCache system. The pool manager also has to decide on which pool the new copy will be created, i.e. staged or p2p-copied.

The behaviour of the pool manager is highly configurable. In order to exploit the full potential of the software it is essential to understand the mechanisms used and how they are configured. The pool manager is a unique cell in the domain "`dCacheDomain`" and consists of several sub-modules: The important ones are the *pool selection unit* (PSU) and the *cost manager* (CM).

The PSU is responsible for finding the pools which the pool manager is allowed to use for a specific transfer-request. From those the CM selects the optimal one. By telling the PSU which pools are permitted for which type of transfer-request, the administrator of the dCache system can adjust the system to any kind of scenario: Separate organizations served by separate pools, special pools for writing the data to a tertiary storage system, pools in a DMZ which serves only a certain kind of data (e.g. for the grid). The following section explains the mechanism employed by the PSU and shows how to configure it with several examples.

The Pool Selection Mechanism

The PSU generates a list of allowable storage-pools for each incoming transfer-request. The PSU-configuration described below tells the PSU which combinations of transfer-request and storage-pool are allowed. Imagine a two-dimensional table with a row for each possible transfer-request and a column for each pool - each field in the table containing either "yes" or "no". For an incoming transfer-request the PSU will return a list of all pools with "yes" in the corresponding row.

Instead of "yes" and "no" the table really contains a *preference* - a non-negative integer. However, the PSU configuration is easier to understand if this is ignored.

Actually maintaining such a table in memory (and as user in a configuration file) would be quite inefficient, because of the many possibilities for the transfer-requests. Instead, the PSU consults a set of rules in order to generate the list of allowed pools. Each such rule is called a *link* because it links a set of transfer-requests to a group of pools. A link consists of a set of condition and a list of pools. If all the conditions are satisfied, the pools belonging to the link are added to the list of allowable pools.

The main task is to understand how the conditions in a link are defined. After we have dealt with that, the preference values will be discussed and a few examples will follow.

The Condition of a Link

The properties of a transfer-request, which are relevant for the PSU, are the following:

Location of the File — The directory of the file in the file system (*perfectly normal file system* - pnfs).

IP Address — The IP address of the requesting host.

Type of Transfer — The type of transfer is either “read”, “write”, or "cache". A request for reading a file which is not stored in the cache, but has to be staged from a connected tertiary storage system will trigger a “cache”-request and a subsequent “read”-request. These will be treated as two separate requests.

The location of the file in the file system is not used directly. Each file has the following two properties which can be set per directory:

Storage Class — The storage class is a string. It is used by a tertiary storage system to decide where to store the file (i.e. on which set of tapes) and dCache can use the storage class for a similar purpose (i.e. on which pools the file can be stored.). A detailed description of the syntax and how to set the storage class of a directory in the PNFS is given in the section called “Storage Classes”.

Cache Class — The cache class is a string with essentially the same functionality as the storage class, except that it is not used by a tertiary storage system. It is used in cases, where the storage class does not provide enough flexibility. It should only be used, if an existing configuration using storage classes does not provide sufficient flexibility

Each link contains one or more *conditions*, all of which have to be satisfied by the transfer-request. Each condition in turn contains several *elementary conditions*. The condition is satisfied if at least one of the elementary conditions is satisfied. For the mathematically inclined this logical structure can be expressed by the following formula:

```
link ==     ( elemCond1 or elemCond2 )
        and ( elemCond3 or elemCond4 or elemCond5 )
        and ... and ( ... ),
```

where the parentheses are the conditions. The first condition contains the elementary conditions `elemCond1` and `elemCond2`, and the second one contains `elemCond3`, `elemCond4`, and `elemCond5`.

There are 3 types of elementary conditions: elementary network (`-net`), storage class (`-store`), and cache class conditions (`-dcache`). Each type imposes a condition on the IP address, the storage class, and the cache class, respectively.

An *elementary network condition* consists of an IP address and a net mask written as “*`IP-address/net mask`*”, say “`111.111.111.0/255.255.255.0`”. It is satisfied, if the request is coming from a host with IP address within the subnet given by the address/netmask pair.

An *elementary storage class condition* is given by a storage class. It is satisfied if the requested file has this storage class. Simple wild cards are allowed: for this it is important to know that a storage class must always contains exactly one @-symbol as will be explained in the section called “Storage Classes”. In an elementary

storage class condition, either the part before the @-symbol or both parts may be replaced by a *-symbol; for example, *@osm and *@* are both valid elementary storage class conditions whereas something@* is invalid. The *-symbol represents a limited wildcard: any string that doesn't contain an @-symbol will match.

An *elementary cache class condition* is given by a cache class. It is satisfied, if the cache class of the requested file agrees with it.

The conditions for the *type of transfer* are not specified with elementary conditions. Instead, each link contains three attributes "-readpref", "-writepref", and "-cachepref", which specify a preference value for the respective types of transfer. If all the conditions in the link are satisfied, the corresponding preference is assigned to each pool the link points to. Since we are ignoring different preference values at the moment, a preference of 0 stands for "no" and a non-zero preference stands for "yes".

The following explanation of the preference values can be skipped at first reading. It will not be relevant, if all non-zero preference values are the same. If you want to try configuring the pool manager right now without bothering about the preferences, you should only use 0 (for "no") and, say, 10 (for "yes") as preferences. The first examples below are of this type.

Preference Values

If more than one preference value different from zero is used, the PSU will not generate a single list but a set of lists, each containing pools with the same preference. The Cost Manager will use the list of pools with highest preference and select the one with the lowest cost for the transfer. Only if all pools with the highest preference are unavailable, the next list will be considered by the Cost Manager. This can be used to configure a set of fall-back pools which are used if none of the other pools are available.

Syntax and Examples

The syntax of the commands for configuring the PSU will be explained with the examples below. These commands can be issued within the PoolManager-cell to change the configuration while the system is running. The **save**-command can then be used to save the current configuration to the file config/PoolManager.conf in the dCache program-directory. This file will be parsed, whenever the dCache system starts up. It is a simple text file containing the corresponding commands. It can therefore also be edited before the system is started. It can also be loaded into a running system with the **reload**-command.

Pool Groups

Pools can be grouped together to pool groups. Consider a host pool1 with two pools, pool1_1 and pool1_2, and a host pool2 with one pool pool2_1. If you want to treat them in the same way, you would create a pool group and put all of them in it:

```
psu create pgroup normal-pools
psu create pool pool1_1
psu addto pgroup normal-pools pool1_1
psu create pool pool1_2
psu addto pgroup normal-pools pool1_2
psu create pool pool2_1
psu addto pgroup normal-pools pool2_1
```

If you later want to treat pool1_2 differently from the others, you would remove it from this pool group and add it to a new one:

```
psu removefrom pgroup normal-pools pool1_2
psu create pgroup special-pools
psu addto pgroup special-pools pool1_2
```

In the following, we will assume that the necessary pool groups already exist. All names ending with "`-pools`" will denote pool groups.

Note that a pool-node will register itself with the pool manager: The pool will be created within the PSU and added to the pool group "`default`", if that exists. This is why the dCache system will automatically use any new pool-nodes in the standard configuration: All pools are in "`default`" and can therefore handle any request.

Separate Write and Read Pools

The dCache we are going to configure receives data from a running experiment, stores the data onto a tertiary storage system, and serves as a read cache for users who want to analyze the data. While the new data from the experiment should be stored on highly reliable and therefore expensive systems, the cache functionality may be provided by inexpensive hardware. It is therefore desirable to have a set of pools dedicated for writing the new data and a separate set for reading.

The simplest configuration for such a setup would consist of two links "write-link" and "read-link". The configuration is as follows:

```
psu create unit -net 0.0.0.0/0.0.0.0
psu create ugroup allnet-cond
psu addto ugroup allnet-cond 0.0.0.0/0.0.0.0

psu create link read-link allnet-cond
psu set link read-link -readpref=10 -writepref=0 -cachepref=10
psu add link read-link read-pools

psu create link write-link allnet-cond
psu set link write-link -readpref=0 -writepref=10 -cachepref=0
psu add link write-link write-pools
```

Why is the condition `allnet-cond` necessary? It is used as a condition which is always true in both links. This is needed, because each link contains at least one condition. The commands contain the words `unit` and `ugroup` for historical reasons. They denote elementary conditions and conditions in our nomenclature.

Restricted Access by IP Address

You might not want to give access to the pools for the whole network, as in the previous example (the section called "Separate Write and Read Pools"), though. Assume, the experiment data is copied into the cache from the hosts with IP `111.111.111.201`, `111.111.111.202`, and `111.111.111.203`. As you might guess, the subnet of the site is `111.111.111.0/255.255.255.0`. Access from outside should be denied. Then you would modify the above configuration as follows:

```
psu create unit -net 111.111.111.0/255.255.255.0
psu create ugroup allnet-cond
psu addto ugroup allnet-cond 111.111.111.0/255.255.255.0

psu create unit -net 111.111.111.201/255.255.255.255
psu create unit -net 111.111.111.202/255.255.255.255
```

```
psu create unit -net 111.111.111.203/255.255.255.255
psu create ugroup write-cond
psu addto ugroup write-cond 111.111.111.201/255.255.255.255
psu addto ugroup write-cond 111.111.111.202/255.255.255.255
psu addto ugroup write-cond 111.111.111.203/255.255.255.255

psu create link read-link allnet-cond
psu set link read-link -readpref=10 -writepref=0 -cachepref=10
psu add link read-link read-pools

psu create link write-link write-cond
psu set link write-link -readpref=0 -writepref=10 -cachepref=0
psu add link write-link write-pools
```

Reserving Pools for Storage and Cache Classes

If pools are financed by one experimental group, they probably do not like it, if it is also used by another group. The best way to restrict data belonging to one experiment to a set of pools is with the help of storage class conditions. If more flexibility is needed, cache class conditions can be used for the same purpose.

Assume, data of experiment A obtained in 2004 is written into subdirectories in the PNFS tree which are tagged with the storage class "`exp-a:run2004@osm`", and similarly for the other years. (How this is done is described in the section called "Storage Classes".) Experiment B uses the storage class "`exp-b:alldata@osm`" for all its data. Especially important data is tagged with the cache class "`important`". (This is described in the section called "Cache Class".) A suitable setup would be

```
psu create ugroup exp-a-cond

psu create unit -store exp-a:run2003@osm
psu addto ugroup exp-a-cond exp-a:run2003@osm
psu create unit -store exp-a:run2004@osm
psu addto ugroup exp-a-cond exp-a:run2004@osm

psu create link exp-a-link allnet-cond exp-a-cond
psu set link exp-a-link -readpref=10 -writepref=10 -cachepref=10
psu add link exp-a-link exp-a-pools

psu create ugroup exp-b-cond

psu create unit -store exp-b:alldata@osm
psu addto ugroup exp-b-cond exp-b:alldata@osm

psu create ugroup imp-cond
psu create unit -dcache important
psu addto ugroup imp-cond important

psu create link exp-b-link allnet-cond exp-b-cond
psu set link exp-b-link -readpref=10 -writepref=10 -cachepref=10
psu add link exp-b-link exp-b-pools

psu create link exp-b-imp-link allnet-cond exp-b-cond imp-cond
psu set link exp-b-imp-link -readpref=20 -writepref=20 -cachepref=20
psu add link exp-b-link exp-b-imp-pools
```

Data tagged with cache class "`important`" will always be written and read from pools in the pool group `exp-b-imp-pools`, except when all pools in this group cannot be reached. Then the pools in `exp-a-pools` will be used.

Note again that these will never be used otherwise. Not even, if all pools in `exp-b-imp-pools` are very busy and some pools in `exp-a-pools` have nothing to do and lots of free space.

The central IT department might also want to set up a few pools, which are used as fall-back, if none of the pools of the experiments are functioning. These will also be used for internal testing. The following would have to be added to the previous setup:

```
psu create link fallback-link allnet-cond
psu set link fallback-link -readpref=5 -writepref=5 -cachepref=5
psu add link fallback-link it-pools
```

Note again that these will only be used, if none of the experiments pools can be reached, or if the storage class is not of the form `exp-a:run2003@osm`, `exp-a:run2004@osm`, or `exp-b:alldata@osm`. If the administrator fails to create the elementary condition `exp-a:run2005@osm` and add it to the condition `exp-a-cond`, the fall-back pools will be used eventually.

Storage Classes

The storage class is a string of the form *`StoreDescriptor@hsm`*, where *`hsm`* denotes the type of tertiary storage system in use, and *`StoreDescriptor`* is a string describing the storage class in a syntax which depends on the used tertiary storage system. If no tertiary storage system is used, it is probably best to use *`hsm`*=osm, since this is tested best. Then the *`StoreDescriptor`* has the syntax *`Store`*:*`StorageGroup`*. These can be set within PNFS per directory. Consider for example the following setup:

```
[root] # cd /pnfs/domain/experiment-a/
[root] # cat ".(tag)(OSMTemplate)"
StoreName myStore
[root] # cat ".(tag)(sGroup)"
STRING
```

This is the setup after a fresh installation and it will lead to the storage class `myStore:STRING@osm`. An adjustment to more sensible values will look like

```
[root] # echo "StoreName exp-a" >! ".(tag)(OSMTemplate)"
[root] # echo "run2004" >! ".(tag)(sGroup)"
```

and will result in the storage class `exp-a:run2004@osm`. To summarize: The storage class will depend on the directory, the data is stored in and is configurable.

Cache Class

Storage classes might already be in use for the configuration of a tertiary storage system. In most cases they should be flexible enough to configure the PSU. However, in rare cases the existing configuration and convention for storage classes might not be flexible enough.

Consider for example a situation, where data produced by an experiment always has the same storage class `exp-a:alldata@osm`. This is good for the tertiary storage system, since all data is supposed to go to the same tape set sequentially. However, the data also contains a relatively small amount of meta-data, which is accessed much more often by analysis jobs than the rest of the data. You would like to keep the meta-data on a dedicated set of dCache pools. However, the storage class does not provide means to accomplish that.

The cache class of a directory is set by the tag `cacheClass` as follows:

```
[root] # echo "metaData" >! ".(tag)(cacheClass)"
```

In the above example the meta-data is stored in directories which are tagged in this way.

There is a nice trick for easy checking of the existing tags in one directory:

```
[root] # grep '' `cat '.(tags)()'`
.(tag)(OSMTemplate):StoreName exp-a
.(tag)(sGroup):run2004
.(tag)(cacheClass):metaData
```

This only works, if the quote-symbols are used correctly. (tick, tick, back-tick, tick, tick, back-tick).

Tags are inherited by sub-directories: Changing a tag of a directory will change the tag of each sub-directory, if the tag has never been changed for this sub-directory directly. Changing tags breaks these inheritance links. Directories in PNFS should never be moved, since this will mess up the inheritance structure and eventually break the whole system.

The Cost Module

From the allowable pools as determined by the *pool selection unit*, the pool manager determines the pool used for storing or reading a file by calculating a *cost* value for each pool. The pool with the lowest cost is used.

If a client requests to read a file which is stored on more than one allowable pool, the *performance costs* are calculated for these pools. In short, this cost value describes how much the pool is currently occupied with transfers.

If a pool has to be selected for storing a file, which is either written by a client or *restored* from a *tape backend*, this performance cost is combined with a *space cost* value to a *total cost* value for the decision. The space cost describes how much it “hurts” to free space on the pool for the file.

The *cost module* is responsible for calculating the cost values for all pools. The pools regularly send all necessary information about space usage and request queue lengths to the cost module. It can be regarded as a cache for all this information. This way it is not necessary to send “get cost” requests to the pools for each client request. The cost module interpolates the expected costs until a new precise information package is coming from the pools. This mechanism prevents clumping of requests.

Calculating the cost for a data transfer is done in two steps. First, the cost module merges all information about space and transfer queues of the pools to calucate the performance and space costs separately. Second, in the case of a write or stage request, these two numbers are merged to build the total cost for each pool. The first step is isolated within a separate loadable class. The second step is done by the cost module.[1]

The Performance Cost

The load of a pool is determined by comparing the current number of active and waiting *transfers* to the maximum number of concurrent transfers allowed. This is done separately for each of the transfer types (*store*, *restore*, pool-to-pool client, pool-to-pool server, and client request) with the following equation:

[1] The next development step will be to add the second calculation as well to the customizable (loadable) class.

perfCost(per Type) = (activeTransfers + waitingTransfers) / maxAllowed .

The maximum number of concurrent transfers (maxAllowed) can be configured with the commands **st set max active** (store), **rh set max active** (restore), **mover set max active** (client request), **p2p set max active** (pool-to-pool server), and **pp set max active** (pool-to-pool client).

Then the average is taken for each mover type where maxAllowed is not zero. For a pool where store, restore and client transfers are allowed, e.g.:

perfCost(total) = (perfCost(store) + perfCost(restore) + perfCost(client)) / 3 ,

and for a read only pool:

perfCost(total) = (perfCost(restore) + perfCost(client)) / 2 .

For a well balanced system, the performance cost should not exceed 1.0.

The Space Cost

In this section only the new scheme for calculating the space cost will be described. Be aware, that the old scheme will be used if the *breakeven parameter* of a pool is larger or equal 1.0.

The cost value used for determining a pool for storing a file depends either on the free space on the pool or on the age of the *least recently used (LRU) file*, which whould have to be deleted.

The space cost is calculated as follows:

If	*freeSpace > gapPara*			then	*spaceCost = 3 * newFileSize / freeSpace*
If	*freeSpace <= gapPara*	and	*lruAge < 60*	then	*spaceCost = 1 + costForMinute*
If	*freeSpace <= gapPara*	and	*lruAge >= 60*	then	*spaceCost = 1 + costForMinute * 60 / lruAge*

where the variable names have the following meanings:

freeSpace — The free space left on the pool

newFileSize — The size of the file to be written to one of the pools, and at least 50MB.

lruAge — The age of the *least recently used file* on the pool.

gapPara — The gap parameter. Default is 4GB. The size of free space below which it will be assumed that the pool is full and consequently the least recently used file has to be removed. If, on the other hand, the free space is greater than `gapPara`, it will be expensive to store a file on the pool which exceeds the free space.

It can be set per pool with the **set gap** command. This has to be done in the pool cell and not in the pool manager cell. Nevertheless it only influences the cost calculation scheme within the pool manager and not the bahaviour of the pool itself.

costForMinute — A parameter which fixes the space cost of a one-minute-old LRU file to *(1 + costForMinute)*. It can be set with the **set breakeven**, where

costForMinute = breakeven * 7 * 24 * 60.

I.e. the the space cost of a one-week-old LRU file will be *(1 + breakeven)*. Not again, that all this only applies if *breakeven < 1.0*

The prescription above can be stated a little differently as follows:

If	freeSpace > gapPara	then	spaceCost = 3 * newFileSize / freeSpace
If	freeSpace <= gapPara	then	spaceCost = 1 + breakeven * 7 * 24 * 60 * 60 / lruAge ,

where `newFileSize` is at least 50MB and `lruAge` at least one minute.

Rationale

As the last version of the formula suggests, a pool can be in two states: Either *freeSpace > gapPara* or *freeSpace <= gapPara* - either there is free space left to store files without deleting cached files or there isn't.

Therefore, `gapPara` should be around the size of the smallest files which frequently might be written to the pool. If files smaller than `gapPara` appear very seldom or never, the pool might get stuck in the first of the two cases with a high cost.

If the LRU file is smaller than the new file, other files might have to be deleted. If these are much younger than the LRU file, this space cost calculation scheme might not lead to a selection of the optimal pool. However, in praxis this happens very seldomly and this scheme turns out to be very efficient.

The Total Cost

The total cost is a linear combination of the *performance* and *space cost*. I.e. totalCost = ccf * perfCost + scf * spaceCost , where `ccf` and `scf` are configurable with the command **set pool decision**. E.g.

```
(PoolManager) admin > set pool decision -spacecostfactor=3 -cpucostfactor=1
```

will give the *space cost* three times the weight of the *performance cost*.

Advanced Customization of the Cost Calculation

The cost calcuation scheme described above can be overwritten by the pool manager "**create**"-option:

```
-costCalculator=newCostCalculator
```

The default value for *`newCostCalculator`* is `CostCalculationV5`. The goal is to compare different pool costs without intermediatly calculating scalar values for performance and space.

Devel

In addition, the PoolCellInfo now is just the CellInfo plus the PoolCostInfo. The WebCollectorV3 has been modified accordingly. The WebCollectorV0 has been removed.

Pool 2 Pool transfer client

The pool 2 pool client transfers are now added to the total cost of a pool and they are reported to the ’pool request’ web page as well.

Although client pool 2 pool transfers seem to be handled within regular queues, they are not. Queuing both, p2p server and queue requests, has a (even though small) probability of deadlocks. So, p2p client requests are never actually queued but they start immediately after they have been requested. The p2p client ’max number of transfers’ is only used to calculate the costs for those transfers.

Chapter 8. The Interface to a Tertiary Storage System

Patrick Fuhrmann

dCache installations, used as a frontend to tertiary storage system, need, at some point, to exchange data this such a system in order to store new, precious files and to retrieve files from the HSM if not yet, or no longer, available on one of the dCache pools. Unfortunately there is no well defined interface for such HSM operations. So the dCache overcomes this problem by calling configurable (dCache external) shell scripts or binaries whenever an HSM store or retrieve operation becomes necessary. The local HSM administrator is responsible for providing this procedure and to make it available and known to the dCache. This small writeup defines the way dCache will call such an external method.

Note

Most dCache distributions do not expect to run together with an HSM backend system. To make this work, make sure that neither in the pool.batch file, nor in the `config/pool.poollist` files the option `lfs=precious` is specified.

Defining the HSM interface.

Each individual pool, which is expected to exchange data with an HSM, has to define a dCache external method to flush/fetch datasets into/from one or more connected HSM's. The command decribed below has either to be given in the command line interface of the corresponding pool while the pool is active (don't forget so "save") or may be added to the pool setup file commands prior to starting the pool.

```
    Syntax : hsm set <hsmName> -command=<fullPathToExternalCommand>

    Example :

     hsm set osm -command=/usr/d-cache/jobs/osm-hsmcp.sh
or
     hsm set enstore -command=/usr/d-cache-deployment/jobs/real-enstore.sh
```

The external method, which might be a shell script or a binary, is called by the dCache with a set of positional arguments (see below). In addition, options may be specified which are appended to the regular argument list on calling the external method.

```
    Syntax : hsm set <hsmName> -<key>=<value>

    Example :

     hsm set osm -command=/usr/d-cache/jobs/osm-hsmcp.sh
     hsm set osm -pnfs=/pnfs/desy.de -somethingElse=true
```

This will result in excuting the following command line whenever a file has to be exchanged with an HSM.

```
/usr/d-cache/jobs/osm-hsmcp.sh put|get <pnfsId> <LocalFilename> \
```

```
        -si=<See Below> \
        -pnfs=/pnfs/desy.de   \
        -somethingElse=true
```

Calling sequence

The external script or binary is launed with 3 positional arguments and at least one option (`-si=<storageInfo>`). Additonial options may follow if defined so with the *hsm set* pool command. Arguments and options are separated by at least on blank character.

```
    Syntax :

        <binary> put|get <pnfsid> <localFileName>   \
               -si=<storageInfo> [more options]
```

The *put*|*get* argument determines the data transfer direction seen from the HSM. *put* means, that data has to be stored into the HSM while *get* means it has be fetched out of the HSM.

The <storageInfo> option is a collection of key value pairs, separated by semicola. All these values are derived from the pnfs database. The possible keys slightly differ, depending on which HSM is addressed. The order of the key value pairs is not determined and may vary between calls. The `-si=` string shouldn't contain blank TAB or newline characters.

Example:

```
    -si=size=1048576000;new=true;stored=false;sClass=desy:cms-sc3;cClass=-;hsm=osm;Host=desy;
```

Table 8.1. Mandatory StorageInfo keys

Key	Meaning
size	Size of the file in bytes
new	*False* if file already in the dCache
stored	*True* if file already stored in the HSM
sClass	HSM depended. Used by the `PoolManager` for pool attraction
cClass	Parent Director tag (`cacheClass`). Used by the `PoolManager` for poolattraction. May be '-'
hsm	Storage Manager name (enstore/osm). Can be overwritten by parent directory tag (`hsmType`).

Table 8.2. Optional StorageInfo keys but used by all HSM's

Key	Meaning
flag-l	Size of the file (if size exceeds 2G)
flag-s	* if file is defined sticky
flag-c	CRC value (currently `1:<hexAdler32>`

Table 8.3. Enstore specific

Key	Meaning
group	Storage Group (e.g. cdf,cms ...)
family	File family (e.g. sgi2test,h6nxl8, ...)
bfid	Bitfile Id (GET only) (e.g. B0MS105746894100000)
volume	Tape Volume (GET only) (e.g. IA6912)
location	Location on tape (GET only) (e.g. : 0000_000000000_0000117)

Table 8.4. OSM specific

Key	Meaning
store	OSM store (e.g. zeus,h1, ...)
group	OSM Storage Group (e.g. h1raw99, ...)
bfid	Bitfile Id (GET only) (e.g. 000451243.2542452542.25424524)

There might be more key values pairs which are used by the dCache internally and which should not affect the behaviour of the hsm copy script.

Table 8.5. Return codes

Return Code	Meaning	Pool Behaviour	
		Into HSM	From HSM
30 <= rc < 40	User defined	Deactivates request	Reports Problem to PoolManager
41	No Space Left on device	Pool Retries	Disables Pool
42	Disk Read I/O Error		Reports Problem to PoolManager
43	Disk Write I/O Error		
All other		Pool Retries	Reports Problem to PoolManager

Special Cases and exceptions

Reading vers. Writing HSM files

When fetching a file from an HSM, the command line contains sufficient information about the location of the dataset within the HSM to get the file. No additional interaction with `pnfs` is needed. So `pnfs` doesn't need to be mounted on read-only pools.

This is different for storing files into an HSM. As a return from the actual HSM put operation, some data has to be stored in `pnfs`. Currently this has to be done directly by the corresponding external HSM script. So, other then for read pools, write pools still need to have `pnfs` mounted.

A future approache will be to transfer the necessay HSM information from the HSM copy script into the dCache using STDOUT. The dCache subsequently performs the necessary `pnfs` store operation through the `PnfsManager`.

Precious files are removed from pnfs

In case a precious file is removed from `pnfs` *before* the hsmcopy-Script (`osmcp.sh` or `real-encp.sh`) is called, the copy on disk is removed and the hsmcopy-Script is not called.

If the file is removed while the hsmcopy-Script is active, the script will encounter an error when writing HSM data into the various `pnfs` layers. In this case it's recommended to return an error code in the 30–39 range to have the request deactivated. So manual intervention is needed to get the situation cleaned up but no attempt is made by the dCache to get the corresponding dataset stored into the HSM again.

Removing files from an backend HSM, triggered by dCache

Whenever a file entry is removed from pnfs (the dCache namespace), dCache takes care that all copies of this file are removed from the various pools. In case, dCache is attached to one or more tertiary storage systems, it provides an interface to allow removing the file from those external systems as well.

As soon as a file entry is removed from `pnfs`, a new file is created within a special, so called, trash directory. (For details, see next paragraph) The name of this newly created file is identical to its inode, resp. pnfsId. A pnfsId is an internal unique identifier for each file within the dCache. PnfsIds don't change if files are renamed and pnfsIds are never reused. The content of the this (new) file is exactly the information written into `level 1` during the HSM store prodecure, discussed in the sections above. This information is usually sufficient to get the file removed from the backend HSM storage system.

The `trash` directory is a local directory residing on the head node, or to be more precise, on the server node where pnfs/chimera is running. In regular dCache installations, the directory is `/opt/pnfsdb/pnfs/trash/1`. After the installation of pnfs, only the path section `/opt/pnfsdb/pnfs/trash` exists. In order to activate the signaling on pnfs file removes, a subdirectory named `1` has to be created within `/opt/pnfsdb/pnfs/trash`. From that point in time, this directory is populated with file entries for each file removed from pnfs/chimera. The mechanism, taking this file information and doing the appropriate HSM specific actions, is resposible for removing those entries if no longer needed, resp. if the file has been removed from the backend HSM.

For exotic dCache installations, the entry `trash=` in `/usr/etc/pnfsSetup` points to the `trash` directory within the local filesystem.

Chapter 9. File Hopping

Patrick Fuhrmann

File hopping is a collective term in dCache, summarizing the possibility of having files being transferred between dCache pools triggered by a variety of conditions. Most of the features described here will be available starting with dCache release 1.6.7. The most prominent examples are:

- If a file is requested by a client but the file resides on a pool from which this client, by configuration, is not allowed to read data, the dataset is transferred to an “allowed” pool first.
- If a pool encounters a steady high load, the system may, if configured, decide to replicate files to other pools to achieve an equal load distribution.
- HSM restore operations may be split into two steps. The first one reads data from tertiary storage to an “HSM connected” pool and the second step takes care that the file is replicated to a general read pool. Under some conditions this separation of HSM and non-HSM pools might become necessary for performance reasons.
- If a dataset has been written into dCache it might become necessary to have this file replicated instantly. The reasons can be, to either have a second, safe copy, or to make sure that clients don’t access the file for reading on the write pools.

File Hopping “on arrival” from outside dCache

“File Hopping on arrival” is a term, denoting the possibility of initiating a pool to pool transfer as the result of a file successfully arriving on a pool. The file must have been written by an external client using any supported protocol (`dCap`, `FTP`, `xrootd`). Files restored from HSM or arriving on a pool as the result of a pool to pool transfer will not yet be forwarded.

Forwarding of incoming files is enabled per pool in the *`hostname`*.poollist file. The pool is requested to send a “replicateFile” message to either the `PoolManager` or to the `HoppingManager`, if available. The different approaches are briefly described below and in more detail in the subsequent sections.

- The “replicateFile” message is sent to the `PoolManager`. This happens for all files arriving at that pool from outside (no restore or p2p). No intermediate `HoppingManager` is needed. The restrictions are
  - All files are replicated. No pre-selection, e.g. on the storage class can be done.
  - The mode of the replicated file is determined by the destination pool and can’t be overwritten. See ’File mode of replicated files’.
- The “replicateFile” message is sent to the `HoppingManager`. The `HoppingManager` can be configured to replicate certain storage classes only and to set the mode of the replicated file according to rules. The file mode of the source file can’t be modified.

File mode of replicated files

The mode of a replicated file can either be determined by settings in the destination pool or by the `HoppingManager`.

- If no `HoppingManager` is used for replication, the mode of the replicated file is determined by the **p2p=*cached|precious*** setting in the *`hostname`*.poollist file of the destination pool. The default setting is **cached**.
- If a `HoppingManager` is used for file replication, the mode of the replicated file is determined by the `HoppingManager` rule responsible for this particular replication. If the destination mode is set to "keep" in the rule, the mode of the destination pool determines the final mode of the replicated file.

File Hopping managed by the PoolManager

File hopping configuration instructs a pool to send a 'replicateFile' request to the `PoolManager` as the result of a file arriving on that pool from some external client. All arriving files will be treated the same. The `PoolManager` will process this request by trying to find a 'link' with the following attributes :

Table 9.1. `PoolManager` Hopping Request Attributes

Data Flow Direction	Protocol	Storage Class	Client IP Number
Pool 2 Pool	dCap/3	Class of file	Configurable

In order to get pool 2 pool enabled for a particular pool, the corresponding entry of that pool in the `XXX.poollist` file has to be extended by the replicateOnArrival key-value pair.

```
PoolName PoolPath  replicateOnArrival=PoolManager,ip-number ...
```

where *`ip-number`* may be a real `IP` number of a farm node which may be taken as example node for others intending to read the file, or the `IP` number may be taken from a non existing `IP` number range. This range can be used to instruct the `PoolManager` to replicate files from this pool to a special set of destination pools.

Please see the section "File mode of replicated files" for the mode of the file on the destination pool.

Example for File Hopping by the PoolManager only

We assume that we want of have all files, arriving on pool `ocean` of host `earth` to be immediately replicated to a subset of read pools. This subset of pools is described by pool group `ocean-copies`. No other pool is member of pool group `ocean-copies`. Other than that, files arriving at pool `mountain` should be replicated to all read pools from which farm nodes on the `131.169.10.0/24` subnet are allowed to read.

The `earth.poollist` file must be modified as followes.

```
ocean     /bigdisk/pools/ocean     replicateOnArrival=PoolManager,192.1.1.1    more options
mountain  /bigdisk/pools/mountain  replicateOnArrival=PoolManager,131.169.10.1 more options
```

While `131.169.10.1` is a legal IP address e.g. of one of you farm nodes, the `192.1.1.1` IP address must not exist anywhere at your site.

Add the following lines to the `PoolManager.conf` in order to instruct the `PoolManager` to replicate files, arriving at the `ocean` pool to be replicated to the `ocean-copies` subset of your read pools.

```
#
#  define the read-pools pool group and add pool members
#
psu create pgroup farm-read-pools
#
psu addto pgroup farm-read-pools read-pool-1
psu addto pgroup farm-read-pools read-pool-2
psu addto pgroup farm-read-pools read-pool-3
psu addto pgroup farm-read-pools read-pool-4

#
psu create unit -net 131.169.10.0/255.255.255.0
#
psu create ugroup farm-network
#
psu addto ugroup farm-network  131.169.10.0/255.255.255.0
#
psu create link farm-read-link any-store any-protocol farm-network
#
psu addto link farm-read-link farm-read-pools
#
psu set link farm-read-link -p2ppref=100 -readpref=100 -writepref=0 -cachepref=XXX...
#
#
#------------------------------------------------------------
#
# create the faked net unit
#
psu create unit -net 192.1.1.1/255.255.255.255
#
psu create ugroup ocean-copy-network
#
psu addto ugroup ocean-copy-network  192.1.1.1/255.255.255.255
#
# we assume that 'any-protocol' and 'any-store' is already defined.
#
psu create link ocean-copy-link any-store any-protocol ocean-copy-network
#
psu addto link ocean-copy-link ocean-copy-pools
#
#  define the ocean-copy pool group and add pool members
#
psu create pgroup ocean-copy-pools
#
psu addto pgroup ocean-copy-pools  read-pool-1
#
psu set link ocean-copy-link -p2ppref=100 -readpref=100 -writepref=0 -cachepref=XXX...
#
#
#
#
```

File Hopping managed by the HoppingManager

Starting the FileHopping Manager service

The `HoppingManager` is not automatically started in 1.7.0. Please perform the following steps to get it started :

- Create a file `hopping.batch` in the `/opt/d-cache/config` directory with the following content :

```
set printout default 3
set printout CellGlue none
onerror shutdown
#
check -strong setupFile
#
copy file:${setupFile} context:setupContext
#
#  import the variables into our $context.
#  don't overwrite already existing variables.
#
import context -c setupContext
#
#   Make sure we got what we need.
#
check -strong serviceLocatorPort serviceLocatorHost
#
create dmg.cells.services.RoutingManager  RoutingMgr
#
#   The LocationManager Part
#
create dmg.cells.services.LocationManager lm \
       "${serviceLocatorHost} ${serviceLocatorPort}"
#
#
create diskCacheV111.services.FileHoppingManager HoppingManager \
     "${config}/HoppingManager.conf -export"
#
```

- change to `/opt/d-cache/jobs`
- Run ./initPackage.sh
- Start the service ./hopping start

Initially no rules are configured for the hopping manager. You may add rules by either edit the `/opt/d-cache/config/HoppingManager.conf` and restart the hopping service, or use the admin interface and save the modifications by 'save' into `HoppingManager.conf`

Configuring pools to use the HoppingManager(x)

In order to instruct the pool to send a 'replicateFile' message to the HoppingManager service, modify the *`hostname`*.poolllist file as follows :

```
...
ocean     /bigdisk/pools/ocean     replicateOnArrival=HoppingManager  more options
...
```

HoppingManager Configuration Introduction

- The `HoppingManager` essentially receives 'replicateFile' messages from pools, configured to support file hopping, and either discards or modifies and forwards them to the `PoolManager`, depending on rules described below.
- The `HoppingManager` decides on the action to perform, based on a set of configurable rules. Each rule has a name. Rules are checked in alphabetic order concerning their names.

- A rule it triggered if the storage class matches the storage class pattern assigned to that rule. If a rule is triggered, it is processed and no further rule checking is performed. If no rule is found for this request the file is not replicated.
- If for whatever reason, a file couldn't be replicated, NO RETRY is being performed.
- Processing a triggered rule can be :
  - The message is discarded. No replication is done for this particular storage class.
  - The rule modifies the 'replicate message', before it is forwarded to the PoolManager.

    The 'destination ip number can be added to the 'replicate file' message. This has the same effect as the ip-number following the "PoolManager" keyword in the *hostname*.poollist file in the 'unconditional replication section above. The rule assignes a 'destination' ip number to the 'replicate message', before it is forwarded to the PoolManager. This has the same effect as the ip-number following the "PoolManager" keyword in the *hostname*.poollist file in the 'unconditional replication section above.

    The mode of the replicated file can be specified. This can either be 'precious', 'cached' or 'keep'. 'keep' means that the pool mode determines the replicated file mode.

    The requested protocol can be specified.

HoppingManager Configuration Reference

```
define hop OPTIONS name pattern precious|cached|keep
   OPTIONS
     -destination=cellDestination # default : PoolManager
     -overwrite
     -continue
     -source=write|restore|*   #  !!!! for experts only      StorageInfoOptions
     -host=destinationHostIp
     -protType=dCap|ftp...
     -protMinor=minorProtocolVersion
     -protMajor=majorProtocolVersion
```

pattern is a storage class pattern which. If the incoming storage class matches this pattern, this rule is processed.

precious|cached|keep determines the mode of the replicated file. 'keep' leaves the destination mode to the pool setting.

destination shouldn't be used. For experts only.

overwrite In case, a rule with the same name already exists, it is overwritten. If this overwrite option is specified, the error will occure.

continue If a rule has triggered and the corresponding action has been performed, no other rules are checked. If the 'continue' option is specified, rule checking continues. This is for debugging purposes only.

source Don't use.

host, protType This is the 'host ip number' and 'protocol type'used by the PoolManager in order find an appropriate pool for the replication request. Please note that 'host' is not the host of the destination pool.

`HoppingManager` configuration examples

Define the `HoppingManager` as destination for the ’replicate file’ requests on the pool(s).

```
...
ocean     /bigdisk/pools/ocean     replicateOnArrival=HoppingManager  more options
...
```

Replicate ’raw’ data files by all experiments.

```
#
define hop replicate-raw  .*:raw@osm -host=Farm Node Ip Number
#
```

Replicate all CMS files to pools assigned to CMS farm nodes and all ATLAS files to pools assigned to ATLAS farm nodes. Don’t replicate any other files.

```
#
define hop replicate-cms    cms:.*@osm    -host=Farm Node Ip Number of cms farm
define hop replicate-atlas  atlase:.*@osm -host=Farm Node Ip Number of atlas farm
#
```

Chapter 10. dCache Partitioning

Patrick Fuhrmann

There are various parameters in the dCache `PoolManager` subsystem, determining the system behaviour on particular events. These are e.g. the cost threshold at which the `PoolManager` initiates pool to pool transfers to smooth the overall system cost, or the balance between performance and disk space related costs to optimize request distribution among data pools. In pre 1.7.0 releases, those parameters were applied to the a whole dCache instance. Starting with 1.7.0 a defined set of parameters (see next section) can have different values for different sections of the complete dCache instance. dCache provides a web page, listing section, assigned parameters and inheritance information.

Parameters, sections and inheritance

Parameters, currently part of the partitioning scheme, are listed within the next paragraph, together with the old and new way of assigning values. A change in the command set has become necessary to reflect the new schema. Each of those parameters may be set to different values for different, so called `dCache sections`. The only `section`, existing without being created, is the `default section`. With the pre 1.7.0 command set, only the `default section` can be manipulated. With the new commmand set, all `sections` can be created or modified. If only a subset of parameters of a `non default section` is defined, the residual parameters of this section are inherited from the `default section`. So, changing a parameter in the `default section`, will change the same parameter of all other sections for which this particular parameter has not been overwritten.

Commands related to dCache partitioning :

- **pm set [*sectionName*] -*parameterName*[=*value*|off]** sets a parameter *parameterName* to a new value. If *sectionName* is omitted, the `default` section is used. If *sectionName* doesn't exist yet, it is (silently) created. If a parameter is set to `off` this parameter is no longer overwritten and is inherited from the `default` section. The `off` doesn't make sense for the `default` section.

- **pm ls [-l] [*sectionName*]** lists a single or all sections. Except for the `default`, only those parameters are shown which are explicitly set. Parameters, not shown, are inherited from the `default` section.

- **pm destroy *sectionName*** destroys a section.

The top menu of the `PoolManager` configuration web pages points to a page summerizing the section status of the system. This is essentially the content of the **pm ls -l**.

List of partitionable parameters

The following list describes which parameters may be used in conjunction with dCache partitioning. The 'Old command set' is still valid for the default parameter section, while new command set has to used to manipulate non-default sections.

Table 10.1. New and old PoolManager parameter names

Old Command	New Command	Parameter Type
set pool decision -spacecostfactor=<value>	pm set [<section>] -spacecostfactor=<value>	float
set pool decision -cpucostfactor=<value>	pm set [<section>] -cpucostfactor=<value>	float
set costcuts -idle=<value>	pm set [<section>] -idle=<value>	float
set costcuts -p2p=<value>	pm set [<section>] -p2p=<value>	float
set costcuts -alert=<value>	pm set [<section>] -alert=<value>	float
set costcuts -halt=<value>	pm set [<section>] -halt=<value>	float
set costcuts -fallback=<value>	pm set [<section>] -fallback=<value>	float
rc set slope <value>	pm set [<section>] -slope=<value>	float
rc set p2p on|off	pm set [<section>] -p2p-allowed	boolean
rc set p2p oncost	pm set [<section>] -p2p-oncost	boolean
rc set p2p fortransfer|notfortransfer	pm set [<section>] -p2p-fortransfer	boolean
rc set stage on|off	pm set [<section>] -stage-allowed	boolean
rc set stage oncost	pm set [<section>] -stage-oncost	boolean
rc set max copies <copies>	pm set [<section>] -max-pnfs-copies=<copies>	integer

Assigning sections to real dCache partitions

A section, so far, is just a set of parameters which may or may not differ from the default set. To let a section relate to a part of the dCache, links are used. Each link may be assigned to exactly one section. If not set, or the assigned section doesn't exist, the link defaults to the `default` section.

```
psu set link linkName -section=sectionName [other link options]
```

Whenever this link is chosen for pool selection, the assosiated parameters of the assigned section will become active for further processing.

Warning

Depending on the way links are setup it may very well happen that more than just one link it triggered for a particular dCache request. This is not illegal but leads to an ambiguity in selecting an appropriate dCache section. If only one of the selected links has a section assigned, this section is chosen. Otherwise, if different links point to different sections, the result is indeterminate. This issue

is not yet solved and we recommend to clean up the PoolManager configuration to eliminate links with the same preferences for the same type of requests.

Examples

For the subsequent examples we assume a basic `PoolManager` setup :

```
#
# define the units
#
psu create -protocol   */*
psu create -protocol   xrootd/*
psu create -net        0.0.0.0/0.0.0.0
psu create -net        131.169.0.0/255.255.0.0
psu create -store      *@*
#
#  define unit groups
#
psu create ugroup  any-protocol
psu create ugroup  any-store
psu create ugroup  world-net
psu create ugroup  xrootd
#
psu addto ugroup any-protocol */*
psu addto ugroup any-store    *@*
psu addto ugroup world-net    0.0.0.0/0.0.0.0
psu addto ugroup desy-net     131.169.0.0/255.255.0.0
psu addto ugroup xrootd       xrootd/*
#
#  define the pools
#
psu create pool poolName
psu create pool ...
#
#  define the pool groups
#
psu create pgroup default-pools
psu create pgroup special-pools
#
psu addto pgroup default-pools poolName
psu addto pgroup default-pools ...
#
psu addto pgrou  specail-pools poolName
psu addto pgroup special-pools ...
```

Disallowing pool to pool transfers for special pool groups based on the access protocol

For a special set of pools, where we only allow the `xrootd` protocol, we don't want the datasets to be replicated on high load while for the rest of the pools we allow replication on hot spot detection.

```
#
#
pm set default        -p2p=0.4
pm set xrootd-section -p2p=0.0
#
psu create link default-link any-protocol any-store world-net
psu add    link default-link default-pools
psu set    link default-link -readpref=10 -cachepref=10 -writepref=0
#
```

```
psu create link xrootd-link xrootd any-store world-net
psu add    link xrootd-link special-pools
psu set    link xrootd-link -readpref=11 -cachepref=11 -writepref=0
psu set    link xrootd-link -section=xrootd-section
#
```

Choosing ’random pool selection’ for incoming traffic only

For the a set of pools we select pools following the default setting of cpu and space related cost factors. For incoming trafic from outside, though, we select the same pools, but in a `randomly` distributed fashion. Please note that this is not really a physical partitioning of the dCache system, but rather a virtual one, applied to the same set of pools.

```
#
#
pm set default          -cpucostfactor=0.2 -spacecostfactor=1.0
pm set incoming-section -cpucostfactor=0.0 -spacecostfactor=0.0
#
psu create link default-link any-protocol any-store desy-net
psu add    link default-link default-pools
psu set    link default-link -readpref=10 -cachepref=10 -writepref=10
#
psu create link default-link any-protocol any-store world-net
psu add    link default-link default-pools
psu set    link default-link -readpref=10 -cachepref=10 -writepref=0
#
psu create link incoming-link any-protocol any-store world-net
psu add    link incoming-link default-pools
psu set    link incoming-link -readpref=10 -cachepref=10 -writepref=10
psu set    link incoming-link -section=incoming-section
#
```

Chapter 11. Central Flushing to tertiary storage systems

Patrick Fuhrmann

This chapter is of interest for dCache instances connected to a tertiary storage system or making use of the mass storage interface for any other reason.

Warning

The central flush control is still in the evaluation phase. The configuration description within this chapter is mainly for the dCache team to get it running on their test systems. The final prodution version will have most of this stuff already be configured.

dCache instances, connected to tertiary storage systems, collect incoming data, sort it by storage class and flush it as soon as certain thresholds are reached. All this is done autonomously by each individual write pool. Consequently those flush operations are coordinated on the level of a pool but not globally wrt a set of write pools or even to the whole dCache instance. Experiences during the last years show, that for various purposes a global flush management would be desirable.

Separation of read/write operations on write pools

The total thoughput of various disk storage systems tend to drop significantly if extensive read and write operations have to be performed in parallel on datasets exceeding the filesystem caches. To overcome this technical obstacle, it would be good if disk storage systems would either allow writing into a pool or flushing data out of a pool into the HSM system, but never both at the same time.

Overcoming HSM limitations and restictions

Some HSM systems, mainly those not coming with their own scheduler, apply certain restrictions on the number of requests being accepted simultaniously. For those, a central flush control system would allow for limiting the number of requests or the number of storage classes being flushed at the same time.

Basic configuration (Getting it to run)

This section describes how to setup a central flush control manager.

- Whitin the `PoolManager`, a pool-group (*`flushPoolGroup`*) has to be created and populated with pools planned to be controlled by the central flush mechanism. An arbitrary number of flush control managers may run within the same dCache instance as long as each can work on its own pool-group and no pool is member of more than one *`flushPoolGroup`*.

- To start the flush control system, an corresponding dCache batch file has to be setup, installed and started. As input parameter, the `HsmFlushControl` cell needs the name of the *`flushPoolGroup`*) and the

name of the driver, controlling the flush behaviour. Within the same batch file more than one flush control manager may be started as long as they get different cell-names and different pool-groups assigned.

- The flush control web pages have to be defined in the `httpd.batch`.

Creating the flush pool group

Creating *`flushPoolGroup`* and adding pools is done within the `config/PoolManager.config` setup file or using the `PoolManager` command line interface. Pools may be member of other pool-groups, as long as those pool-groups are not managed by other flush control managers.

```
psu create pool pool-1
psu create pool ...
#
psu create pgroup flushPoolGroup
#
psu addto pgroup flushPoolGroup  pool-1
psu addto pgroup flushPoolGroup  ...
#
```

Creating and activating the hsmcontrol batch file

```
#
set printout default errors
set printout CellGlue none
#
onerror shutdown
#
check -strong setupFile
#
copy file:${setupFile} context:setupContext
#
import context -c setupContext
#
check -strong serviceLocatorHost serviceLocatorPort
#
create dmg.cells.services.RoutingManager  RoutingMgr
#
create dmg.cells.services.LocationManager lm \
     "${serviceLocatorHost} ${serviceLocatorPort}"
#
create diskCacheV111.hsmControl.flush.HsmFlushControlManager FlushManagerName  \
        "flushPoolGroup  \
         -export   -replyObject \
         -scheduler=SchedulerName  \
         Scheduler specific options \
        "
#
```

Which the following meaning of the variables :

- *`flushPoolGroup`* needs to be the name of the pool group defined in the `PoolManager.conf` files.
- *`SchedulerName`* is the name of a class implementing the `diskCacheV111.hsmControl.flush.HsmFlushSchedulable` interface.
- *`Scheduler specific options`* may be options specific to the selected scheduler.

Initially there are three schedulers available :

- `diskCacheV111.hsmControl.flush.driver.HandlerExample` may be used as an example implementation of the HsmFlushScheduler interface. The functionality is useless in an production environment but can be useful to check the functionality of the central flush framework. If one allows this driver to take over control it will initiate the flushing of data as soon as it becomes aware of it. One the other hand it supports a mode where is doesn't do anything except preventing the individual pools from doing the flush autonomously. In that mode, the driver assumes the flushes to be steered manually by the flush web pages decribed in the next paragraph. The latter mode is enabled by starting the flush driver with the *`Scheduler specific options`* set to `-do-nothing`

- `diskCacheV111.hsmControl.flush.driver.AlternateFlush` is intended to provide suffient functionality to cope with issues described in the introduction of the paragraph. Still quite some code and knowledge has to go into this driver.

- `diskCacheV111.hsmControl.flush.driver.AlternatingFlushSchedulerV1` is certainly the most useful driver. It can be configured to flush all pools on a single machine simultaniously. It is trigger by space consumption, number of files within a pool or the time the oldest file resides on a pool without having been flushed. Please checkout the next section for details on configuration and usage.

The AlternatingFlushSchedulerV1 driver

The AlternatingFlushSchedulerV1 is an alternating driver, which essentially means that it either allows data to flow into a pool, or data going from a pool onto an HSM system but never both at the same time. Data transfers from pools to other pools or from pools to clients are not controlled by this driver. In order to minimize the latter one should configure HSM write pools to not allow transfers to clients but doing pool to pool transfers first.

Configuration

```
#
create diskCacheV111.hsmControl.flush.HsmFlushControlManager FlushManagerName  \
        "flushPoolGroup  \
         -export   -replyObject \
         -scheduler=diskCacheV111.hsmControl.flush.driver.AlternatingFlushSchedulerV1  \
         -driver-config-file=${config}/flushDriverConfigFile \
        "
#
```

Where *`flushPoolGroup`* is a PoolGroup defined in the `PoolManager.conf` file, containing all pools which are intended to be managed by this FlushManager. *`flushDriverConfigFile`* is a file within the dCache `config` directory holding property values for this driver. The driver reloads the file whenever it changes its modification time. One should allow for a minute of two before new setting are getting activated. The configuration file has to contain key value pairs, separated by the = sign. Keys, not corresponding to a driver property are silently ignored. Properties, not set in the configuration file, are set to some reasonable default value.

Properties

Driver properties may be specified by a configuration file as described above or by talking to the driver directly using the command line interface. Driver property commands look like :

```
driver properties -PropertyName=value
```

Because the communication with the driver is asynchronous, this command will never return an error. To check if the new property value has been accepted by the driver, run the sequence

```
driver properties
info
```

It will list all available properties together with the currently active values.

Table 11.1. Driver Properties

Property Name	Default Value	Meaning
max.files	500	Collect this number of files per pool, before flushing
max.minutes	120	Collect data for this amount of minutes before flushing
max.megabytes	500 * 1024	Collecto this number of megabytes per pool before flushing
max.rdonly.fraction	0.5	Do not allow more than this percentage of pools to be set read only
flush.atonce	0	Never flush more than that in one junk
timer	60	Interval timer (minimum resolution)
print.events	false	Print events delivered by the FlushManager
print.rules	false	Print remarks from the rule engine
print.poolset.progress	false	Print progress messages

The selection process

Finding all flush candidates

A pool is becoming a flush candidate if either the number of files collected exceeds `max.files` or the number of megabytes collected exceeds `max.megabytes` or the oldest file, not flushed yet, is becoming older than `max.minutes`.

Selecting the best candidate

Pool Candidates are sorted according to a metric, which is essentially the sum of three items. The number of files devided by `max.files`, the number of megabytes devided by `max.megabytes` and the age of the oldest file devided by `max.minutes`.

The pool with the highest metric is chosen first. The driver determines the hardware unit, this pools resides on. The intention is to flush all pools of this unit simultaniously. Depending on the configuration, the unit can be either a disk partition or a host. After the hardware unit is determined, the driver adds the number of pools on that unit to the number of pools already in ’read only’ mode. If this sum exceeds the total number of pools in the flush pool

group, multiplied by the `max.rdonly.fraction` property, the pool is NOT selected. The process proceeds until a pool, resp. a hardware unit complies with these contrains.

The hardware unit, a pool belongs to, is set by the 'tag.hostname' field in the `config/`*`hostname`* file.

The actual flush process

If a pool is flushed, all storage groups of that pool are flushed, and within each storage group all precious files are flushed simultaniously. Setting the property `flush.atonce` to some positive nonzero number will advise each storage group not to flush more than this number of files per flush operation. There is no way to stop a flush operation which has been triggered by the FlushManager. The pool will proceed until all files, belonging to this flush operation, have been successfully flushed or failed to flush. Though, the next section describes how to suspend the flush pool selection mechanism.

Suspending and resuming flush operations

The driver can be advised to suspend all new flush operations and switch to halt mode.

```
driver command suspend
```

To resume flushing :

```
driver command resume
```

In suspend mode, all flushing is halted which sooner or later results in overflowing write pools.

Driver interactions with the flush web portal or the GUI

Flush Manager operations can be visualized by configuring the flush web pages, described in one of the subsequent sections or by using the flush module of the 'org.pcells' GUI. In addition to monitoring, both mechanisms allow to set the pool I/O mode (rdOnly, readWrite) and to flush individual storage groups or pools. The problem may be that those manual interactions interfere with driver operations. The AlternatingFlushSchedulerV1 tries to cope with manual interactions as follows :

- The pool I/O mode may be manually set to `read only` while the pool is not flushing data and therefor naturally would be in read write mode. If this pool is then subsequently chosen for flushing, and the flushing process has finished, the pool is NOT set back to readWrite mode, as it usually would be, but it stays in readOnly mode, because the driver found this mode when starting the flush process and assumes that it had been in that mode for good reason. So, setting the pool I/O mode to readOnly while the pool is not flushing freezes this mode until manually changed again. Setting the I/O mode to readOnly while the pool is flushing, has no effect.
- If a pool is in readOnly mode because the driver has been initiating a flush process, and the pool is manually set back to readWrite mode, is stays in readWrite mode during this flush process. After the flush sequence has finished, the pool is set back to normal as if no manual intervention had taken place. It does *not* stay with readWrite mode forever as it stays in readOnly mode forever in the example above.

When using the web interface or the GUI for flushing pools or individual storage groups, one is responsible for setting the pool I/O mode oneself.

Setting up and using the flush control web pages.

In order to keep track on the flush activities the flush control web pages need to be activated. Add a new `set alias` directive somewhere between the **define context httpdSetup endDefine** and the **endDefine** command in the `/opt/d-cache/config/httpd.batch` file.

```
define context httpdSetup endDefine
...
set alias flushManager class diskCacheV111.hsmControl.flush.HttpHsmFlushMgrEngineV1
 mgr=FlushManagerName
...
endDefine
```

Additional flush managers may just be added to this command, separated by commas. After restarting the 'httpd' service, the flush control pages are available at `http://headnode:2288/flushManager/mgr/*`.

The flush control web page is split into 5 parts. The top part is a switchboard, pointing to the different flush control managers installed. (listed in the mgr= option of the **set alias flushManager** in the `config/httpd.config`). The top menu is followed by a `reload` link. Its important to use this link instead of the 'browsers' reload button. The actual page consists of tree tables. The top one presents common configuration information. Initially this is the name of the flush cell, the name of the driver and whether the flush controller has actually taken over control or not. Two action buttons allow to switch between centrally and locally controlled flushing. The second table lists all pools managed by this controller. Information is provided on the pool mode (readonly vers. readwrite), the number of flushing storage classes, the total size of the pool and the amount of precious space per pool. Action buttons allow to toggle individual pools between `ReadOnly` and `ReadWrite` mode. Finally the third table presents all storage classes currently holding data to be flushed. Per storage class and pool, characteristic properties are listed, like total size, precious size, active and pending files. Here as well, an action button allows to flush individual storage classes on individual pools.

Warning

The possibilty to interactively interact with the flush manager needs to be supported by the driver choosen. Please check the information on the individual driver how far this is supported.

Examples

Configuring Central Flushing for a single Pool Group with the AlternatingFlushSchedulerV1 driver

Setting up the PoolManager configuration

Add all pools, which are planned to be centrally flushed to a PoolGroup, lets say `flushPoolGroup` :

```
psu create pool migration-pool-1
psu create pool migration-pool-2
#
psu create pgroup flushPoolGroup
#
psu addto pgroup flushPoolGroup migration-pool-1
psu addto pgroup flushPoolGroup migration-pool-2
#
```

Setting up the central flush batch file.

Create a batchfile `/opt/d-cache/config/hsmcontrol.batch` with the following content :

```
#
set printout default 3
set printout CellGlue none
#
onerror shutdown
#
check -strong setupFile
#
copy file:${setupFile} context:setupContext
#
import context -c setupContext
#
check -strong serviceLocatorHost serviceLocatorPort
#
create dmg.cells.services.RoutingManager  RoutingMgr
#
create dmg.cells.services.LocationManager lm \
     "${serviceLocatorHost} ${serviceLocatorPort}"
#
create diskCacheV111.hsmControl.flush.HsmFlushControlManager FlushManager  \
        "flushPoolGroup \
         -export   -replyObject \
         -scheduler=diskCacheV111.hsmControl.flush.driver.AlternatingFlushSchedulerV1
  \
          -driver-config-file=${config}/flushPoolGroup.conf \
        "
#
```

Change to `/opt/d-cache/jobs` and run `./initPackage.sh`. Ignore possible warnings and error messages. The Script will create the necessary links, mainly the `jobs/hsmcontrol` startup file. To start the central service run

```
cd /opt/d-cache/jobs
./hsmcontrol start
```

This setup will produce quite some output in `/var/log/hsmcontrol.log`. Reduce the output level if this is not required.

```
set printout default errors
```

Setting up the driver properties file

Create a file in `/opt/d-cache/config` named `flushPoolGroup.conf` with the content listed below. You may change the content any time. The driver will reload it after awhile.

```
#
#  trigger parameter
#
max.files=4
max.minutes=10
max.megabytes=200
#
#  time interval between rule evaluation
#
timer=60
#
# which fraction of the pool set should be rdOnly (maximum)
#
max.rdonly.fraction=0.999
#
#  output steering
#
print.events=true
print.rules=true
print.pool.progress=true
print.poolset.progress=true
mode=auto
```

Chapter 12. gPlazma authorization in dCache

Ted Hesselroth

`gPlazma` is a cell in dCache that authorizes users. Cells make requests to `gPlazma` by submitting user credential information to it, receiving the authorization decision and site-specific user information such as uid, gid, and rootpath in return.

The acronym stands for Grid-aware PLuggable AuthoriZation Management, and supports the use of plugins which implement various selectable authorization methods. The four currently-available methods are:

- kpwd : This is the “legacy” method. The `dcache.kpwd` file is used to map a user’s DN to a local username, and the same file is used in a second mapping of the username to the uid, gid, and rootpath. As in all methods, if the mappings succeed, file system access is done using the obtained uid and gid, and a check is done that the local path of the transfer starts with the designated rootpath.
- grid-mapfile : This method employs a grid mapfile. From the mapfile, the user’s DN is mapped to a username. A second file, storage-authzdb, is used for the mapping of the username to the uid, gid, and rootpath.
- gplazmalite-vorole-mapping : In this method the mapping to the username is done from the concatenation of the user’s DN with the user’s Role (or, more precisely, with the user’s Fully Qualified Attribute Name). The mapping of username to uid, gid, and rootpath is through the storage-authzdb file.
- saml-vo-mapping : The DN and Role are mapped to a username via a callout to a GUMS [http://grid.racf.bnl.gov/GUMS/] server. The GUMS service may run an extension which returns the uid, gid, and rootpath as well. Otherwise, the mapping of username to uid, gid, and rootpath is through the storage-authzdb file.

The following describes how to use gPlazma in dCache.

Installation

`gPlazma` is included in dCache version 1.7 or higher. As of that version, the `gPlazma` cell can be called from `GridFTP` and `GSIdCap` doors and the `SRM` server.

For the dCache 1.7 version, there must be host certificates on the node running the `gPlazma` cell. For version 1.8, host certificates are needed if the option to delegate (see the section called “Delegation to `gPlazma`”) to `gPlazma` is used.

Depending on which authorization methods are to be used, some configuration files must be modified. The configuration files described here must exist on the node on which you wish to run the `gPlazma` cell and must contain the correct site-specific information for the dCache on which it is deployed.

Configuring the gPlazma Policy File

The `gPlazma` policy file, located in `${ourHomeDir}/etc/dcachesrm-gplazma.policy`, controls which authorization plugins will be tried and the order in which they will be tried. The first of these is specified lines containing "ON" or "OFF" for each plugin, for example

```
# Switches
xacml-vo-mapping="OFF"
saml-vo-mapping="ON"
kpwd="ON"
grid-mapfile="OFF"
gplazmalite-vorole-mapping="OFF"
```

The order is specified by assigning a different number to each plugin, such as

```
# Priorities
xacml-vo-mapping-priority="5"
saml-vo-mapping-priority="1"
kpwd-priority="3"
grid-mapfile-priority="4"
gplazmalite-vorole-mapping-priority="2"
```

In the above example, the saml-vo-mapping plugin would be tried first. If authorization was denied for that method, or if the authentication method itself failed, then the kpwd plugin would be tried. The "Priorities" numbering shows that if gplazmalite-vorole-mapping were to also be turned on, it would be tried after the saml-vo-mapping plugin and before the kpwd method.

Having more than one plugin turned on allows a plugin to be used as fallback for another plugin that may fail. It also allows for the authorization of special users who may be denied by the other methods.

The policy file also contains a section for each of the plugins, for configuration specific to that plugin. These sections are described in the documentation for each plugin, as follows.

Configuring the kpwd Plugin

The section in the gPlazma policy file for the kpwd plugin specifies the location of the `dcache.kpwd` file, for example

```
# dcache.kpwd
kpwdPath="/opt/d-cache/etc/dcache.kpwd"
```

To maintain only one such file, make sure that this is the same location as defined in dCacheSetup.

Please see dCache documentation for dcache.kpwd [http://www.dcache.org/downloads/Release.notes1.6.5-2] for how to create this file.

Configuring the grid-mapfile Plugin

Two file locations are defined in the policy file for this plugin:

```
# grid-mapfile
gridMapFilePath="/etc/grid-security/grid-mapfile"
storageAuthzPath="/etc/grid-security/storage-authzdb"
```

Preparing the grid mapfile

The grid mapfile is the same as that used in other applications. It can be created in various ways, either by connecting directly to VOMS or GUMS servers, or by hand.

Each line contains two fields: a DN (Certificate Subject) in quotes, and the username it is to be mapped to.

```
"/DC=org/DC=doegrids/OU=People/CN=Gina Carlson 584065" uscms01
```

When using the grid-mapfile plugin, the `storage-authzdb` file must also be configured. See the section called “`storage-authzdb`” for details.

storage-authzdb

In `gPlazma`, except for the dcache.kpwd plugin, authorization mapping is a two-step process. First, a username is obtained from a mapping of the user’s DN or DN and role, then a mapping of username to uid, gid, rootpath is performed. The `storage-authzdb` file is used for the second mapping.

Preparing storage-authzdb

The location of storage-authzdb is `/etc/grid-security/storage-authzdb`. The file must contain a line specifying the version of the storage-authdb format.

```
version 2.1
```

The other lines in the file each contain eight fields: the string "authorize", followed by the username, read-write permission, uid, gid, and three paths.

```
authorize uscms01 read-write 60076 5063 / /pnfs/fnal.gov/resilient/uscms01 /
```

In the storage-authzdb file, there must be a line for each username to be authorized. The existence of three paths is for legacy puposes. The second path is of most importance; it is the path under which the user is allowed to access files. It is permissible to simply use a “`/`” for the second path.

```
authorize uscms01 read-write 60076 5063 / / /
```

in which case the user will be authorized for any path (the filesystem permissions in pnfs must also allow the transfer).

The first path is nearly always left as “`/`”, but it may be used as a home directory in interactive session, as a subdirectory of the second path. Upon login, the second path is used as the user’s root, and a “cd” is performed to the first path. The first path is always defined as being relative to the second path.

Starting with dCache 1.9, multiple gids can be assigned by using comma-separated values for the GID file, as in

```
authorize uscms01 read-write 60076 5063,5071,6843 / / /
```

The lines of the storage-authzdb file are similar to the “login” lines of the `dcache.kpwd` file. If you alreay have a `dcache.kwpd` file, create storage-authzdb by taking the lines from your `dcache.kpwd` file that start with the word "login", for example,

```
login uscms01 read-write 60076 5063 / /pnfs/fnal.gov/resilient/uscms01 /
```

and replacing the word “login” with “authorize”.

Support for the Priority Field in storage-authzdb

In the future, dCache services may support the use of priorites, to be assigned in storage-authzdb. To assign priorities in storage-authzdb, replace the stated version number with "2.2"

```
version 2.2
```

In the remainder of the file, the fourth field of each line is the priority, which is an integer. Otherwise the fields have the same definitions as in version 2.1.

```
authorize uscms01     read-write    2       60076 5063   / /pnfs/fnal.gov/resilient/uscms01 /
authorize cmssoft     read-write    0       60501 5502   / /pnfs/fnal.gov/reduction/cmssoft /
```

Using version 2.1, the default priority is “0”, therefore use this value if it is desired to have the same behavior as in 2.2. Interpretation of the priority value is dependent on the implementation of any dCache service which may use it, however, the convention is that higher numerical values of the field result in higher priority. See the documentation of the specific service in question for details.

There are currently no dCache services which make use of the priority field.

Configuring the gplazmalite-vorole-mapping Plugin

The `gPlazma` policy file contains two lines for this plugin.

```
# Built-in gPLAZMAlite grid VO role mapping
gridVoRolemapPath="/etc/grid-security/grid-vorolemap"
gridVoRoleStorageAuthzPath="/etc/grid-security/storage-authzdb"
```

The second is the storage-authz-db used in other plugins. See the above documentation Configuring storage-authzdb for how to create the file.

Preparing `grid-vorolemap`

The file is similar in format to the grid-mapfile, however there is an additional field following the DN (Certificate Subject), containing the FQAN (Fully Qualified Attribute Name).

```
"/DC=org/DC=doegrids/OU=People/CN=Gina Carlson 584065" "/cms/uscms" uscms01
"/DC=org/DC=doegrids/OU=People/CN=Gina Carlson 584065" "/cms/uscms/Role=cmsprod" cmsprod
```

Therefore each line has three fields: the user’s DN, the user’s FQAN, and the username that the DN and FQAN combination are to be mapped to.

The FQAN is sometimes semantically referred to as the “role”. The same user can be mapped to different usernames depending on what their role is. The role is determined by how the user creates their proxy, for example, using **voms-proxy-init**. The FQAN contains the user’s Group, Role (optional), and Capability (optional). The latter two may be set to the string “NULL”, in which case they will be ignored by the plugin.

If a user is authorized in multiple roles, for example

```
"/DC=org/DC=doegrids/OU=People/CN=Gina Carlson 584065" "/cms/uscms" uscms01
"/DC=org/DC=doegrids/OU=People/CN=Gina Carlson 584065" "/cms/uscms/Role=cmsuser" cms2847
"/DC=org/DC=doegrids/OU=People/CN=Gina Carlson 584065" "/cms/uscms/Role=cmsphedex" phedex
"/DC=org/DC=doegrids/OU=People/CN=Gina Carlson 584065" "/cms/uscms/Role=cmsprod" cmsprod
```

they would be mapped to the username corresponding to the role found in the proxy that the user creates for use by the client software. Starting in dCache 1.8, if the user actually creates several roles in their proxy, authorization (and subsequent check of path and filesystem permissions) will be attempted for each role in the order that they are found in the proxy. In a griftp URL, the user may also explicity request a username

```
gsiftp://cmsprod@griddoor1.oursite.edu:2811/testfile1
```

in which case other roles will be disregarded.

Authorizing a VO

Instead of individual DNs, it is allowable to use `*` or `"*"` as the first field, such as

```
"*" "/cms/uscms/Role=cmsprod" cmsprod
```

In that case, any DN with the corresponding role will match. It should be noted that a match is first attempted with the explicit DN. Therefore if both DN and `"*"` matches can be made, the DN match will take precedence. This is true for the revocation matches as well (see below).

Thus a user with subject

```
"/DC=org/DC=doegrids/OU=People/CN=Ted Hesselroth 897321"
```

and role

```
"/cms/uscms/Role=cmsprod"
```

will be mapped to username `cmsprod` via the above storage-authzdb line with `"*"` for the DN, except if there is also a line such as

```
"/DC=org/DC=doegrids/OU=People/CN=Ted Hesselroth 898521" "/cms/uscms/Role=cmsprod" uscms01
```

in which case the username will be `uscms01`.

Revocation Entries

To create a revocation entry, add a line with `-` as the username, such as

```
"/DC=org/DC=doegrids/OU=People/CN=Timur Perelmutov 623542" "/uscms/production" -
```

or modify the username of the entry if it already exists. The behaviour is undefined if there are two entries which differ only by username.

Since DN is matched first, if a user would be authorized by his VO membership through a `"*"` entry, but is matched according to his DN to a revocation entry, authorization would be denied. Likewise if a whole VO were denied in a revocation entry, but some user in that VO could be mapped to a username through his DN, then authorization would be granted.

More Examples

Suppose that there are users in production roles that are expected to write into the storage system data which will be read by other users. In that case, to protect the data the non-production users would be given read-only access. Here in `/etc/grid-security/grid-vorolemap` the production role maps to username `cmsprod`, and the role which reads the data maps to `cmsuser`.

```
"*" "/cms/uscms/Role=cmsprod" cmsprod
"*" "/cms/uscms/Role=cmsuser" cmsuser
```

The read-write privilege is controlled by the third field in the lines of `/etc/grid-security/storage-authzdb`

```
authorize cmsprod  read-write  9811 5063 / /pnfs/fnal.gov/data /
authorize cmsuser  read-only  10001 6800 / /pnfs/fnal.gov/data /
```

Another use case is when users are to have their own directories within the storage system. This can be arranged within the `gPlazma` configuration files by mapping each user's DN to a unique username and then mapping each username to a unique root path. As an example, lines from `/etc/grid-security/grid-vorolemap` would therefore be written

```
"/DC=org/DC=doegrids/OU=People/CN=Selby Booth" "/cms" cms821
"/DC=org/DC=doegrids/OU=People/CN=Kenja Kassi" "/cms" cms822
"/DC=org/DC=doegrids/OU=People/CN=Ameil Fauss" "/cms" cms823
```

and the corresponding lines from `/etc/grid-security/storage-authzdb` would be

```
authorize cms821 read-write 10821 7000 / /pnfs/fnal.gov/data/cms821 /
authorize cms822 read-write 10822 7000 / /pnfs/fnal.gov/data/cms822 /
authorize cms823 read-write 10823 7000 / /pnfs/fnal.gov/data/cms823 /
```

Starting with dCache 1.8, regular expressions are supported in the `/etc/grid-security/storage-authzdb` file. Substitutions by regular expression group are also permitted. Place a regular expression in the username field of the `storage-authzdb` file. Any groups in the regular expression (defined by enclosure in parentheses) can be referred to in later fields of the line, and the corresponding susbstitution will be made when the file is read. For example, the above lines for granting users individual directories can be replaced with

```
authorize cms(\d\d\d) read-write 10$1 7000 / /pnfs/fnal.gov/data/cms$1 /
```

in which case `cms821` matches `cms(\d\d\d)` and the group `(\d\d\d)` is substituted in `10$1` to yield `10821` and in `/pnfs/fnal.gov/data/cms$1` to yield `/pnfs/fnal.gov/data/cms821`, and so on.

Configuring the saml-vo-mapping Plugin

There are two lines in the policy file for this plugin.

```
# SAML-based grid VO role mapping
mappingServiceUrl="https://gums.oursite.edu:8443/gums/services/GUMSAuthorizationServicePort"
# Time in seconds to cache the mapping in memory
saml-vo-mapping-cache-lifetime="60"
```

The first line containins the URL for the GUMS web service. Replace the URL with that of the site-specific GUMS [http://grid.racf.bnl.gov/GUMS/]. When using the "GUMSAuthorizationServicePort", the service will only provide the username mapping and it will still be necesary to have the storage-authzdb file used in other plugins. See the above documentation Configuring storage-authzdb for how to create the file. If a GUMS server providing a "StorageAuthorizationServicePort" with correct uid, gid, and rootpath information for your site is available, the storage-authzdb file is not necesary.

The second line contains the value of the caching lifetime. In order to decrease the volume of requests to the SAML authorization (GUMS) service, authorizations for the saml-vo-mapping method are by default cached for a period of time. To change the caching duration, modify the saml-vo-mapping-cache-lifetime value in `/opt/d-cache/etc/dcachesrm-gplazma.policy`

```
saml-vo-mapping-cache-lifetime="120"
```

To turn off cach caching, set the value to 0.The default value is 60 seconds except for in dCache version 1.9.2, in which the default value is 0; caching is turned off by default in that version.

Configuring the xacml-vo-mapping Plugin

Beginning with dCache version 1.9.2, gPlazma includes a new authorization plugin, to support the XACML authorization schema. Using XACML with SOAP messaging allows gPlazma to acquire authorization mappings from any service which supports the obligation profile for grid interoperability [http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=2952]. Servers presently supporting XACML mapping are the latest releases of GUMS and SCAS. Using the new plugin is optional, and previous configuration files are still compatible with gPlazma. If the installation is an upgrade it will change /opt/d-cache/config/gPlazma.batch. It is normally not necessary to change this file, but if you have customized the previous copy, transfer your changes to the new batch file.

The configuration is very similar to that for the saml-vo-mapping plugin. There are two lines for the configuration.

```
# XACML-based grid VO role mapping
XACMLmappingServiceUrl="https://fledgling09.fnal.gov:8443/gums/services/GUMSXACMLAuthorizationServicePort"
# Time in seconds to cache the mapping in memory
xacml-vo-mapping-cache-lifetime="180"
```

for a GUMS [http://grid.racf.bnl.gov/GUMS/] server, or, for an SCAS server,

```
# XACML-based grid VO role mapping
XACMLmappingServiceUrl="https://scas.europeansite.eu:8443"
# Time in seconds to cache the mapping in memory
xacml-vo-mapping-cache-lifetime="180"
```

As for the saml-vo-mapping, the first line containins the URL for the web service.Replace the URL with that of the site-specific GUMS or SCAS server. When using the "GUMSXACMLAuthorizationService-Port" (notice the difference in service name from that for the saml-vo-mapping) with a GUMS server, the service will only provide the username mapping and it will still be necesary to have the storage-authzdb file used in other plugins. See the above documentation Configuring storage-authzdb for how to create the file. An SCAS server will return a UID, a primary GID, and secondary GIDS, but not a rootpath. A storage-authzdb file will be necesary to assign the rootpath. Since SCAS does not return a username, the convention in `gPlazma` is to use "uid:gid" for the username, where uid is the string representation of the uid returned by SCAS, and gid is the string representation of the primary GID returned by SCAS. Thus a line such as

```
authorize 13160:9767 read-write 13160 9767 / /pnfs/fnal.gov/data /
```

in `/etc/grid-security/storage-authzdb` will serve to assign the user mapped by SCAS to uid=13160 and primary gid=9767 the rootpath `/pnfs/fnal.gov/data`. It is best for consistency's sake to fill in the UID and GID fields with the same values as in the "uid:gid" field. Additional secondary gids can be assigned by using comma-separated values in the GID field. Any gids there not already returned as secondary gids by SCAS will be added to the secondary gids list.

The second line contains the value of the caching lifetime. In order to decrease the volume of requests to the XACML authorization (GUMS or SCAS) service, authorizations for the saml-vo-mapping method are by default cached for a period of time. To change the caching duration, modify the saml-vo-mapping-cache-lifetime value in `/opt/d-cache/etc/dcachesrm-gplazma.policy`

```
saml-vo-mapping-cache-lifetime="120"
```

To turn off cach caching, set the value to 0. For xacml-vo-mapping, the default value is 0; caching is turned off by default.

An example policy file

Here is an example of how a policy file might be set up.

```
saml-vo-mapping="ON"
kpwd="ON"
grid-mapfile="OFF"
gplazmalite-vorole-mapping="OFF"
saml-vo-mapping-priority="1"
kpwd-priority="3"
grid-mapfile-priority="4"
gplazmalite-vorole-mapping-priority="2"
kpwdPath="/opt/d-cache/etc/dcache.kpwd"
gridMapFilePath="/etc/grid-security/grid-mapfile"
storageAuthzPath="/etc/grid-security/storage-authzdb"
mappingServiceUrl="https://fledgling09.fnal.gov:8443/gums/services/GUMSAuthorizationServicePort"
saml-vo-mapping-cache-lifetime="60"
gridVoRolemapPath="/etc/grid-security/grid-vorolemap"
gridVoRoleStorageAuthzPath="/etc/grid-security/storage-authzdb"
```

In this case, gPlazma will attempt to authorize first through a GUMS server, and fall back to using dcache.kpwd. The mappingServiceUrl would have to be changed to a GUMS server appropriate for the site.

The Setup Files

Changes to Setup files require a restart of the cell.

The gPlazmaSetup File

This file will normally be a link to ${ourHomeDir}/config/dCacheSetup. The section of the file which controls the operation of the cell is as follows:

```
gplazmaPolicy=${ourHomeDir}/etc/dcachesrm-gplazma.policy
#
# gPlazmaNumberOfSimutaneousRequests  30
# gPlazmaRequestTimeout               30
#
# useGPlazmaAuthorizationModule=false
# useGPlazmaAuthorizationCell=true
```

The first line defines the path to the gPlazma policy file. It is set to the default location of the policy file. If the path to the gPlazma policy file changes, enter the full path in the above line.

The variable gPlazmaNumberOfSimutaneousRequests determines the number of threads which will be started on the gPlazma cell to handle requests. The default value should be sufficient, but may be raised or lowered depending on the capacity of the hardware and other processes running on it.

The variable gPlazmaRequestTimeout is the amount of time in seconds that a request thread has to finish an authorization decision. If this time is exceeded, authorization is denied.

The next two variables are not used by gPlazma, but by GridFTP door and SRM. Their description follows.

The gridftpdoorSetup and srmSetup Files

Thes two files will normally be links to ${ourHomeDir}/config/dCacheSetup. The lines of interest for gPlazma are

```
gplazmaPolicy=${ourHomeDir}/etc/dcachesrm-gplazma.policy
```

and

```
# useGPlazmaAuthorizationModule=false
# useGPlazmaAuthorizationCell=true
```

The latter two lines control whether the GridFTP door or SRM will authorize locally, or use the gPlazma for authorization. The default is to use the gPlazma cell for authorization. If both values are set to false, the GridFTP door or SRM will use the dcache.kpwd lookup method. A dcache.kpwd file must be

present on the GridFTP door or SRM node in that case. It is possible to use gPlazma methods on the GridFTP door or SRM without calling the gPlazma cell. See the following section.

Using Direct Calls of gPlazma Methods

Cells may also call gPlazma methods as an alternative, or as a fallback, to using the gPlazma cell.

Operation without a gPlazma Cell

If the gPlazma cell is not started, other cells can still authorize by calling gPlazma methods directly from a pluggable module. The gPlazma control files and host certificates are needed on the node from which authorization will take place. To invoke the gPlazma modules, modify the following line in gridftp-doorSetup or srmSetup to

```
useGPlazmaAuthorizationModule=true
```

and make sure that the gplazmaPolicy line defines a valid gPlazma policy file on the node for which authorization is to occur:

```
gplazmaPolicy=${ourHomeDir}/etc/dcachesrm-gplazma.policy
```

No adjustable timeout is available, but any blocking would likely be due to a socket read in the saml-vo-mapping plugin, which is circumvented by a built-in 30-second timeout.

Using a gPlazma Cell with a Direct-Call Fallback

Both a call to the gPlazma cell and the direct call of the gPlazma module may be specified. In that case, authentication will first be tried via the gPlazma cell, and if that does not succeed, authentication by direct invocation of gPlazma methods will be tried. Modify the following lines to:

```
useGPlazmaAuthorizationModule=true
useGPlazmaAuthorizationCell=true
```

Make sure that the line for gplazmaPolicy

```
gplazmaPolicy=${ourHomeDir}/etc/dcachesrm-gplazma.policy
```

set to a local policy file on the node. The gPlazma policy file on the GridFTP door or SRM does not have to specify the same plugins as the gPlazma cell.

gPlazma Options

Validating User Attributes in dCache 1.8

A user's roles (Fully Qualified Attribute Names) are read from the certificate chain found within the proxy. These attributes are signed by the user's VOMS server when the proxy is created. Starting with version 1.8,

gPlazma supports checking the signature of the attributes against VOMS server certificates installed on the gPlazma node. To have gPlazma validate the proxy attributes, place the voms server certificates in /etc/grid-security/vomsdir and in /opt/d-cache/etc/dcachesrm-gplazma.policy set

```
vomsValidation="true"
```

The default is false. The same would need to be done on the SRM or any door node for which gPlazma modules are called directly as a fallback.

Validating User Attributes in dCache 1.9

In version 1.9, VOMS attribute validation in gPlazma uses a method in which installation of the voms server certificate is not required. Instead the signature on an attribute is checked against the ca certificate that signed the voms server certificate. To have gPlazma validate the proxy attributes in dCache 1.9, write configuration directories and "*.lsc" files in /etc/grid-security/vomsdir for each authorized voms server according to these instructions [https://twiki.cern.ch/twiki/bin/view/LCG/VomsFAQforServiceManagers] and in /opt/d-cache/etc/dcachesrm-gplazma.policy set

```
vomsValidation="true"
```

As with previous versions, the default is false. Whether validation is on or not, there must be a non-empty /etc/grid-security/vomsdir on the node which is running gPlazma. It is enough to do

```
[root] # mkdir /etc/grid-security/vomsdir
touch /etc/grid-security/vomsdir/empty-cert.pem
```

to create the non-empty directory.

Delegation to gPlazma

In dCache version 1.7, SRM or the GridFTP door delegated the user's credentials to gPlazma, and the user's attributes were extracted from the secure context. For performance purposes, starting in version 1.8 the delegation step is not performed. To turn on delegation, in /opt/d-cache/config/dCacheSetup on the node running SRM or the GridFTP door, set the line

```
delegateToGPlazma=true
```

The default value is false. To support delegation, host certificates must exist on the host which runs gPlazma.

Chapter 13. dCache as xRootd-Server

Martin Radicke

This chapter explains how to configure dCache in order to access it via the `xrootd` protocol, allowing `xrootd`-Clients like ROOT's TXNetfile and xrdcp to do file operations against a dCache instance in a transparent manner. The current implementation in dCache 1.7.0 is based on the most recent production version of the `xrootd` protocol (2.4.5).

Setting up

The xrootd functionality is contained in all dCache releases starting from 1.7.0. Versions prior to this are not supported!

To allow file transfers in and out of dCache using xrootd, a new `xrootd door` must be started. This door acts then as the entry point to all `xrootd` requests. Compared to the native xrootd server-implementation (produced by SLAC), the `xrootd door` refers to the `redirector node`.

To enable the `xrootd door`, just change the config file `${dCacheHome}/etc/node_config` so that it contains the line

```
..  XROOTD=yes
      ..
```

After a restart of the dCache core-services, done by executing

```
[root] # ${dCacheHome}/bin/dcache-core restart
```

the `xrootd` door should be running. A few minutes later it should appear at the web monitoring interface under "Cell Services" (see the section called “The Web Interface for Monitoring dCache”).

Warning

Starting from version 1.7.0 (patchlevel 20), the `xrootd door` is set to readonly by default. In prior versions, the door is started in unsecure mode, giving clients unrestricted read and write access. Please refer to the section called “`xrootd` security” on how to setup security.

Parameters

The default port the `xrootd door` is listening on is 1094. This can be changed in `${dCacheHome}/config/dCacheSetup` by setting the variable "xrootdPort" to the desired value (again restart required).

The number of parallel `xrootd` file transfers per pool node is limited by the portrange defined in `${dCacheHome}/config/dCacheSetup`, since each transfer occupies one (not firewalled) port for its own. The portrange can be set via the property "`org.dcache.net.tcp.portrange`" in the line

```
..
```

```
java_options="-server -Xmx512m -XX:MaxDirectMemorySize=512m -Dorg.globus.tcp.port.range=50000,52000
-Dsun.net.inetaddr.ttl=1800 -Djava.net.preferIPv4Stack=true -Dorg.dcache.dcap.port=0
-Dorg.dcache.net.tcp.portrange=33115:33145 "
..
```

In the above example, the maximum would be 30 simultaneous `xrootd` transfers per pool. Any change to the door port or the portrange requires a dDache-core- or accordingly dCache-pool- restart.

Quick tests

The subsequent paragraphs describe a quick guide on how to test `xrootd` using the `xrdcp` and `ROOT` clients.

Copying files with xrdcp

A simple way to get files in and out of dCache via `xrootd` is the command xrdcp. It is included in every xrootd and ROOT distribution.

To transfer a single file in and out of dCache, just issue

```
[user] $ xrdcp /bin/sh root://door_hostname//pnfs/site.de/data/xrd_test
      [user] $ xrdcp root://door_hostname//pnfs/site.de/data/xrd_test /dev/null
```

Accessing files from within ROOT

This simple ROOT example shows how to write a randomly filled histogram to a file in dCache:

```
root [0] TH1F h("testhisto", "test", 100, -4, 4);
root [1] h->FillRandom("gaus", 10000);
root [2] TFile *f = new TXNetFile("root://door_hostname//pnfs/site.de/data/test.root","new");
061024 12:03:52 001 Xrd: Create: (C) 2004 SLAC INFN XrdClient 0.3
root [3] h->Write();
root [4] f->Write();
root [5] f->Close();
root [6] 061101 15:57:42 14991 Xrd: XrdClientSock::RecvRaw: Error reading from socket: Success
061101 15:57:42 14991 Xrd: XrdClientMessage::ReadRaw: Error reading header (8 bytes)
```

Closing remote `xrootd` files that live in dCache produces this warning, but has absolutely no effect on subsequent ROOT commands. It happens because dCache closes all `TCP` connections after finishing a file transfer, while xrootd expects to keep them open for later resue.

To read it back into ROOT from dCache:

```
root [7] TFile *reopen = TXNetFile ("root://door_hostname//pnfs/site.de/data/test.root","read");
root [8] reopen->ls();
TXNetFile**             //pnfs/site.de/data/test.root
 TXNetFile*             //pnfs/site.de/data/test.root
  KEY: TH1F     testhisto;1     test
```

xrootd security

Read-Write access

Per default dCache xrootd is restricted to read-only, because plain xrootd is completely unauthenticated. A typical error message on the clientside if the server is read-only looks like:

```
 [user] $ xrdcp -d 1 /bin/sh root://ford.desy.de//pnfs/desy.de/data/xrd_test2
Setting debug level 1
061024 18:43:05 001 Xrd: main: (C) 2004 SLAC INFN xrdcp 0.2 beta
061024 18:43:05 001 Xrd: Create: (C) 2004 SLAC INFN XrdClient kXR_ver002+kXR_asyncap
061024 18:43:05 001 Xrd: ShowUrls: The converted URLs count is 1
061024 18:43:05 001 Xrd: ShowUrls: URL n.1: root://ford.desy.de:1094//pnfs/desy.de/data/asdfas.
061024 18:43:05 001 Xrd: Open: Access to server granted.
061024 18:43:05 001 Xrd: Open: Opening the remote file /pnfs/desy.de/data/asdfas
061024 18:43:05 001 Xrd: XrdClient::TryOpen: doitparallel=1
061024 18:43:05 001 Xrd: Open: File open in progress.
061024 18:43:06 5819 Xrd: SendGenCommand: Server declared: Permission denied. Access is read only.
(error code: 3003)
061024 18:43:06 001 Xrd: Close: File not opened.
Error accessing path/file for root://ford//pnfs/desy.de/data/asdfas
```

To enable read-write access, edit the following line in ${dCacheHome}/config/dCacheSetup

```
..
xrootdIsReadOnly=false
..
```

and do a restart of the dCache core services.

Please note that due to the unauthenticated nature of this access mode, files can be written and read to/from any subdirectory in the pnfs namespace (including the automatic creation of parent directories). Because there is no user information at the time of request, new files/subdirectories generated through xrootd will inherit UID/GID from its parent directory.

Permitting write access on selected pnfs directories

To overcome the security issue of uncontrolled xrootd write access mentioned in the previous section, it is possible to restrict write access on a per-directory basis (including subdirectories). This feature is available from dCache 1.7.0-36 on.

To activate this feature, a colon-seperated list containing the full pnfs paths of authorized directories must be provided in ${dCacheHome}/config/dCacheSetup:

```
..
xrootdAllowedPaths=/pnfs/site.de/path1:/pnfs/site.de/path2
..
```

A restart of the xrootd door is required to make the changes take effect. As soon as *${xrootdAllowedPaths}* is set, all write requests to directories not matching the allowed path list will be refused.

Token-based authorization

The `xrootd` dCache implementation includes a generic mechanism to plug in different authorization handler. The only plugin available so far implements token-based authorization as suggested in http://people.web.psi.ch/feichtinger/doc/authz.pdf.

The first thing to do is to setup the keystore. The keystore file basically specifies all RSA-keypairs used within the authorization process and has exactly the same syntax as in the native xrootd tokenauthorization implementation. In this file, each line beginning with the keyword `KEY` corresponds to a certain Virtual Organisation (VO) and specifies the remote public (owned by the file catalogue) and the local private key belonging to that VO. A line containing the statement `"KEY VO:*"` defines a default keypair that is used as a fallback solution if no VO is specified in token-enhanced `xrootd` requests. Lines not starting with the `KEY` keyword are ignored. A template can be found in `${dCacheHome}/etc/keystore.template`.

The keys itself have to be converted into a certain format in order to be loaded into the authorization plugin. dCache expects both keys to be binary DER-encoded (Distinguished Encoding Rules for ASN.1). Furthermore the private key must be PKCS #8-compliant and the public key must follow the X.509-standard.

The following example demonstrates how to create and convert a keypair using OpenSSL:

```
Generate new RSA private key
[root] # openssl genrsa -rand 12938467 -out key.pem 1024

Create certificate request
[root] # openssl req -new -inform PEM -key key.pem -outform PEM -out certreq.pem

Create certificate by self-signing certificate request
[root] # openssl x509 -days 3650 -signkey key.pem -in certreq.pem -req -out cert.pem

Extract public key from certificate
[root] # openssl x509 -pubkey -in cert.pem -out pkey.pem
[root] # openssl pkcs8 -in key.pem -topk8 -nocrypt -outform DER -out new_private_key
[root] # openssl enc -base64 -d -in pkey.pem -out new_public_key
```

Only the last two lines are performing the actual conversion, therefore you can skip the previous lines in case you already have a keypair. Make sure that you keystore file correctly points to the converted keys.

To enable the plugin, it is necessary to uncomment and customize the following two lines in the file `${dCacheHome}/config/dCacheSetup`, so that it looks like

```
..
 xrootdAuthzPlugin=org.dcache.xrootd.security.plugins.tokenauthz.TokenAuthorizationFactory
 xrootdAuthzKeystore=Path_to_your_Keystore
 ..
```

After doing a restart of dCache-core, any requests without an appropriate token should result in an error saying "`authorization check failed: No authorization token found in open request, access denied.(error code: 3010)`".

If both tokenbased authorization and read-only access are activated, the read-only restriction will dominate (local settings have precedence over remote file catalogue permissions).

Precedence of security mechanisms

The previously explained methods to restrict access via `xrootd` can also be used in conjunction. The precedence applied in that case is as following:

The permission check executed by the authorization plugin (if one is installed) is given the lowest priority, because it can controlled by a remote party. E.g. in the case of tokenbased authorization, access control is determined by the file catalogue (global namespace).

To allow local site's administrators to override remote security settings, write access can be further restricted to few directories (based on the local namespace, the `pnfs`). Setting `xrootd` access to read-only has the highest priority, overriding all other settings.

Chapter 14. dCache Storage Resource Manager

Gerd Behrmann
Dmitry Litvintsev
Timur Perelmutov
Vladimir Podstavkov

Introduction

Storage Resource Managers (SRMs) are middleware components whose function is to provide dynamic space allocation and file management on shared storage components on the Grid. SRMs support protocol negotiation and a reliable replication mechanism. The SRM specification standardizes the interface, thus allowing for a uniform access to heterogeneous storage elements.

General SRM Concepts

SRM interface consists of the five categories of functions: Space Management, Data Transfer, Request Status, Directory and Permission Functions. SRM interface utilizes Grid Security Infrastructure (GSI) for authentications. SRM service is a Web Service implementation of a published WSDL document. Please visit SRM Working Group Page [http://sdm.lbl.gov/srm-wg/] to see the SRM Version 1.1 and SRM Version 2.2 protocol specification documents.

SURLs

SRM defines a protocol name SRM, and introduces a way to address the files stored in the SRM managed storage by Site URL of the format `srm://<host>:<port>/[<web service path>?SFN=]<path>`. Examples of the Site URLs a.k.a. SRM URLs are:

```
srm://fapl110.fnal.gov:8443/srm/managerv2?SFN=//pnfs/fnal.gov/data/test/file1,
srm://fapl110.fnal.gov:8443/srm/managerv1?SFN=/pnfs/fnal.gov/data/test/file2
srm://srm.cern.ch:8443/castor/cern.ch/cms/store/cmsfile23
```

All SRM functions that operate on files use Site URLs (SURLs) for file references.

Data Transfer functions

There are three functions for performing data transfers in SRM, namely srmPrepareToGet, srmPrepareToPut and srmCopy. These are SRM Version 2.2 names, in SRM Version 1.1 they were called get, put and copy, but their roles were essentially the same. These functions take list of sources(srmPrepareToGet), destinations(srmPrepareToPut) or both (srmCopy). The role of the srmPrepareToGet function is to prepare the system for the receipt of the data into the given file names, to make sure that the system has enough space to store the files, that the user has sufficient privileges to create the files in the paths designated by the SURLs. The purpose of the srmPrepareToPut function is to prepare the data stored in files, designated by the

given SURLs , that are already a part of the system for the network access; SRM needs again to check that the user has sufficient privileges to access the files in the paths designated by the SURLs. One of the features of thesrmPrepareToGet and srmPrepareToPut functions is that they both support transfer protocol negotiation. This means that in case of both of these functions client supplies a list of supported transfer protocols and SRM server computes the Transfer URL in the first protocol from the list that it supports. Depending on the implementation the real action in the Storage System performed in response to this invocation may range from simple SURL to TURL translation to a Stage from Tape to Disk Cache and dynamic selection of the transfer host and transfer protocol depending on the protocol availability and current load on each of the transfer server load. It is a responsibility of the client to perform the transfer and to notify the SRM that it is done with the files. srmCopy function performs a copy between a local and a remote storage system, it is given a list of source – destination URL pairs. At least one of the URLs in each pair must be an SURL of file in the SRM system contacted with the srmCopy request. Second URL can be a local or remote SURL or URL in some other transfer protocol. In case of srmCopy the SRM system performs data transfer itself, without data ever flowing though the client's computer.

The Data Transfer functions are asynchronous, initial SRM call leads to the start of the execution of the client's request, and the functions return the request statuses, that contain unique request tokens, that can be used in subsequent calls for periodic polling of the status of the request. Once the SRM completes the requests, and clients are done with the data transfers, clients notify the system that they are done with the files and are ready to release the associated resources, the client notifies the system by execution of the srmReleaseFiles in case of srmPrepareToGet or srmPutDone in case of srmPrepareToPut. In case of srmCopy, system knows when the transfer is competed and resources can be released, so it requires no special function at the end.

Clients are free to cancel the requests at any time by execution of the srmAbortFiles or srmAbortRequest.

Space Management functions

SRM Version 2.2 introduces a concept of space reservation. Space reservation is a promise by the storage system to make certain amount of storage space of certain type available for usage for a specified period of time. Space reservation is made using srmReserveSpace function. In case of successful reservation, a unique name, called space token is assigned to the reservation. Space token can be used during the transfer operations to tell the system to put the files being manipulated or transferred into an associated space reservation . A storage system ensures that the reserved amount of the disk space is indeed available, thus providing a guarantee that a client does not run out of space until all space promised by the reservation has been used. When files are deleted, the space is returned to the space reservation.

A space reservation has a property called retention policy. Possible values of retention policy are Replica, Output and Custodial. The retention policy describes the quality of the storage service that will be provided for the data (files) stored in this space reservation. Replica corresponds to the lowest quality of the service, usually associated with storing a single copy of each file on the disk. Custodial is the highest quality service, usually interpreted as storage of the data on Tape. WLCG has decided not to use Output retention policy in its data grid. Output is an intermediate retention policy is stronger than Replica and weaker than Custodial, and in dCache Output retention policy will possibly be used for files managed by Resilient Manager, which will make several internal copies of each file, distributed on distinct instances of hardware. Once a file is written into a given space reservation, it inherits the reservation's retention policy.

Another property of the space reservation is called access latency. The two values allowed are Nearline and Online. Nearline means that the data stored in this reservation are allowed to be stored in such a way that retrieving them might require storage system to perform additional preparatory steps (staging data from tape to a disk cache for example). Online means that data is readily available and it will not take long to start

reading the date. In case of dCache Online means that there will always be a copy of the file on disk, while Nearline does not provide such guarantee. As with retention policy, once a file is written into a given space reservation, it inherits the reservation's access latency.

DCache however only manages write space, i.e. only space on disk can be reserved and only for write operations. Once files are migrated to tape, and if no copy is required on disk, space used by these files is returned back into space reservation. When files are read back from tape and cached on disk, they are not counted as part of any space. SRM Space reservation can be assigned a non-unique description, then the description cab be used in the future to discover all space reservation with a given description.

Properties of the SRM Space Reservations can be discovered using SrmGetSpaceMetadata function. Space Reservations might be released with srmReleaseSpace. For a complete description of the available functions please see SRM Version 2.2 Specification [http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html].

Utilization of the Space Reservations for Data Storage

SRM Version 2.2 srmPrepareToPut and srmCopy pull mode transfers allow the user to specify a space reservation token or a retention policy and an access latency. In the protocol, any of these values are optional, and it is up to the implementation to decide what to do, if these properties are not specified. The specification does however require that if a space reservation is given, then any access latency or retention policy specified must match the same properties of the space reservation.

Directory functions

Starting from SRM Version 2.2, interface provides a complete set of the directory management functions. These are srmLs, srmRm, srmMkDir, srmRmDir and srmMv.

Permission functions

SRM Version 2.2 support the following three space permission functions, srmGetPermission, srmCheckPermission and srmSetPermission. dCache contains a rudimentary implementation of these functions that mostly allow setting and checking of the Unix file permission.

SRM Service

dCache SRM is implemented as Web Service Interface running under Apache Tomcat application server and Axis Web Services engine. This service starts a dCache SRM domain with a main SRM cell and a number of other cells SRM service relies on. These are SrmSpaceManager, PinManager, RemoteGsiftpCopyManager, etc. Of these services only SRM and SrmSpaceManager require special configuration.

dCache specific concepts

Link Groups

dCache 1.8 PoolManager supports new type of objects called LinkGroups. Each link group corresponds to a number of dCache pools in the following way: LinkGroup is a collection of the Links, each of which is a collection of the PoolGroups associated (Linked, hence a name "Link") with a set of the Pool Selection Units or PSUs. Each link group knows about its available size, which is a sum of all available sizes in all

the pools included in this link group. In addition link group has 5 boolean properties called replicaAllowed, outputAllowed, custodialAllowed, onlineAllowed and nearlineAllowed, the values of these properties (true or false) can be configured in `PoolManager.conf`.

Space Reservations

In dCache 1.8 each SRM Space Reservation is made against the total available disk space of a particular link group. The total space in dCache that can be reserved is the sum of the available sizes of all Link Groups. If dCache is configured correctly each byte of disk space, that can be reserved, belongs to one and only one Link Group. Therefore it is important to make sure that no pool belongs to more than one pool group, no Pool Group belongs to more than one Link and no Link belongs to more than one LinkGroup.

Files written into a space made within a particular link group will end up on one of the pools referred to by this link group. The difference between the Link Group's available space and the sum of all the current space reservation sizes is the available space in the link group.

Explicit and Implicit Space Reservations for Data Storage in dCache

In dCache, if a space reservation is specified, the file will be stored in it (assuming the user has permission to do so in the name space).

If the reservation token is not specified, and implicit space reservation is enabled, then a space reservation will be performed implicitly for each SRM v1.1 and SRM 2.2 srmPrepareToPut or srmCopy in pull mode. If an Access Latency and a Retention Policy are specified, the user defined retention policy and default access latency. If the user has not specified Access Latency or Retention Policy (or if SRM v1.1 is used) , the system will attempt to extract special tags (not surprisingly called "AccessLatency" and "RetentionPolicy") from PNFS namespace from the directory to which file is being written. If the tags are present, then their values will determine the default Access Latency or Retention Policy that will be used for implicit space reservations. If the tags are not present, then system wide defaults will be used. If no implicit space reservation can be made, the transfer will fail. (Note: some clients also have default values, which are used when not explicitly specified by the user. I this case server side defaults will have no effect.)

If the implicit space reservation is not enabled in dCache 1.8 the pools in the link groups will be excluded from consideration and only the remaining pools will be considered to be the candidates for storing the incoming data, and classical pool selection mechanism will be used. If the space reservation is not used and no LinkGroups are specified, the system behavior will be exactly the same as in dCache 1.7.

Space Manager access control

When SRM Space Reservation request is executed, its parameters, such as reservation size, lifetime, access latency and retention policy as well as user's Virtual Organization (VO) membership information is forwarded to the SRM SpaceManager.

Space Manager uses a special file for listing all the Virtual Organizations (VOs) and all the VO Roles that are permitted to make reservations in the given link group. List of the allowed VOs and VO Roles, together with the total available space and replicaAllowed, outputAllowed, custodialAllowed, onlineAllowed and nearlineAllowed properties of the group is than matched against the information from the user request in order to determine if a given space reservation can be made in particular link group. Once a space reservation

is created, no access control is performed, any user can attempt to store the files in this space reservation, provided he or she knows the exact space token.

Choosing The right hardware and OS for the SRM node

Hardware

We recommend to install dCache SRM server on a separate node with sufficient memory and a fast disk optimized for database application. For example Fermilab US-CMS T1 site uses the following hardware for SRM node. Dual Intel Xeon Duo, 4 GB RAM, 3ware raid disk array.

Operating System

Latest Scientific Linux or RHEL would do.

The kernel.shmmax=1073741824 and kernel.shmall=1073741824 kernel parameters should be set for a 4GB RAM Machine. This can be accomplished by running:

```
[root] # echo 'kernel.shmmax=1073741824' >>  /etc/sysctl.conf
[root] # echo 'kernel.shmall=1073741824' >>  /etc/sysctl.conf
[root] # /bin/sysctl -p
```

The exact content of US-CMS T1 SRM `sysctl.conf` is:

```
kernel.core_uses_pid = 1
kernel.sysrq = 1
kernel.panic = 60
fs.file-max = 131072
net.ipv4.ip_forward = 0
vm.vfs_cache_pressure = 10000
# Keep this amount of memory free for emergency, IRQ and atomic allocations.
vm.min_free_kbytes = 65535
# Network tune parameters
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_sack = 0
net.ipv4.tcp_window_scaling = 1
kernel.shmmax=1073741824
kernel.shmall=1073741824
```

Configuring Postgres Database

Install the latest PostgreSQL database from PostgreSQL web site [http://www.postgresql.org/download/]. While some like RPMs, others find that they have 100% guarantee of compatibility of the software only if they build it locally from sources. In later case source rpms or archive of sources are available.

We highly recommend to make sure that PostgreSQL database files are stored on a separate disk that is not used for anything else (not even PostgreSQL logging). BNL Atlas Tier one observed a great improvement in srm-database communication performance after they deployed postgres on a separate dedicated machine.

To provide seamless local access to the database please make the following modifications:

The file `/var/lib/pgsql/data/pg_hba.conf` should contain the following lines

```
local   all         all                           trust
host    all         all         127.0.0.1/32      trust
host    all         all         ::1/128           trust
```

If SRM or srm monitoring is going to be installed on a separate node, you need to add entry for this node as well:

```
host    all         all       monitoring node    trust
host    all         all       srm node    trust
```

The `postgresql.conf` should contain the following:

```
#to enable network connection on the default port
max_connections = 100
port = 5432
...
shared_buffers = 114688
...
work_mem = 10240
...
#to enable autovacuuming
stats_row_level = on
autovacuum = on
autovacuum_vacuum_threshold = 500  # min # of tuple updates before
                                   # vacuum
autovacuum_analyze_threshold = 250      # min # of tuple updates before
                                        # analyze
autovacuum_vacuum_scale_factor = 0.2    # fraction of rel size before
                                        # vacuum
autovacuum_analyze_scale_factor = 0.1   # fraction of rel size before
#
# setting vacuum_cost_delay might be useful to avoid
# autovacuum penalize general performance
# it is not set in US-CMS T1 at Fermilab
#
# In IN2P3 add_missing_from = on
# In Fermilab it is commented out

# - Free Space Map -
max_fsm_pages = 500000

# - Planner Cost Constants -
effective_cache_size = 16384            # typically 8KB each
```

To enable dCache SRM components access to the database server with the user srmdcache:

```
[root] # createuser -U postgres --no-superuser --no-createrole --createdb --pwprompt srmdcache
```

SRM will use the database dcache for storing its state information:

```
[root] # createdb -U srmdcache dcache
```

Configuring SRM Domain

Once database and and JVM are installed and database is running, you may install dCache SRM.

Install dCache server.rpm

```
[root] # rpm -Uvh dcache.server.rpm
```

node_config

Copy /pt/d-cache/etc/node_config.template into /opt/d-cache/etc/node_config

Edit /opt/d-cache/etc/node_config

```
NODE_TYPE=custom
...
SRM=yes
...
# all other parameters should be turned off on "srm only" node
```

srm_setup.env

Edit /opt/d-cache/etc/srm_setup.env

- Make sure that JAVA_HOME is set to correct value, for example

```
JAVA_HOME=/usr/java/jdk1.5.0_07
```

- Tomcat port does not interfere with with services that are already using network

```
TOMCAT_PORT=8080
```

- If you are going to run the monitoring on the same node:

```
TOMCAT_HTTP_ENABLED=true
JAVA_OPTS="-Xmx512m -Djava.awt.headless=true"
```

install dCacheSetup

Copy /opt/d-cache/etc/dCacheSetup.template into /opt/d-cache/config/dCacheSetup and edit it so that, serviceLocatorHost and serviceLocatorPort point to central dcache node:

```
serviceLocatorHost=host of central node
serviceLocatorPort=11111
```

following SRM parameters should be configured as following:

```
srmVacuum=false
srmDbName=dcache
srmDbUser=srmdcache
```

Make sure that both `srmCopyReqThreadPoolSize` and `remoteGsiftpMaxTransfers` are set to the same values and the common value should be the roughly equal to the maximum number of the SRM -to -SRM copies your system can sustain. So if you think about 3 gridftp transfer per pool and you have 30 pools than the number should be 3x30=90.

```
srmCopyReqThreadPoolSize=90
remoteGsiftpMaxTransfers=90
```

Note US-CMS T1 has:

```
srmCopyReqThreadPoolSize=2000
remoteGsiftpMaxTransfers=2000
```

Tomcat/axis deployment

Run

```
[root] # /opt/d-cache/install/install.sh
```

Starting and stopping SRM domain

Run

```
[root] # /opt/d-cache/bin/dcache-core start
```

to start SRM domain.

Run

```
[root] # /opt/d-cache/bin/dcache-core stop
```

to stop SRM domain.

SRM Logs

SRM might produce a lot of logs, especially if it run in debug mode. Need to run SRM in debug mode is greatly reduced if SRM monitoring is installed. It is recommended to make sure that logs are redirected into a file on large disk. dCache SRM 1.7 logs into `/opt/d-cache/libexec/apache-tomcat-5.5.20/logs/catalina.out`.

SRM configuration for experts

There are a few parameters in `dCacheSetup` that you might find useful for nontrivial SRM deployment.

srmSpaceManagerEnabled

`srmSpaceManagerEnabled` tells if the space management is activated in SRM.

Possible values are `yes` and `no`. Default is `yes`.

Usage example:

```
srmSpaceManagerEnabled=yes
```

srmImplicitSpaceManagerEnabled

`srmImplicitSpaceManagerEnabled` tells if the space should be reserved for `SRM` Version 1 transfers and for `SRM` Version 2 transfers that have no space token specified. Will have effect only if srmSpaceManagerEnabled.

Possible values are `yes` and `no`. This is enabled by default, disabled if `srmSpaceManagerEnabled` is set to `no`.

Usage example:

```
srmImplicitSpaceManagerEnabled=yes
```

overwriteEnabled

`overwriteEnabled` tells to `SRM` and `GridFTP` servers if the overwrite is allowed. If enabled on `SRM` node, should be enabled on all `GridFTP` nodes.

Possible values are `yes` and `no`. Default is `no`.

Usage example:

```
overwriteEnabled=yes
```

srmOverwriteByDefault

`srmOverwriteByDefault` Set this to `true` if you want overwrite to be enabled for `SRM` v1.1 interface as well as for `SRM` v2.2 interface when client does not specify desired overwrite mode. This option will be considered only if `overwriteEnabled` is set to `yes`.

Possible values are `true` and `false`. Default is `false`.

Usage example:

```
srmOverwriteByDefault=false
```

srmDatabaseHost

`srmDatabaseHost` tells to `SRM` which database host to connect to. Do not change unless you know what you are doing.

Default value is `localhost`.

Usage example:

```
srmDatabaseHost=database-host.example.org
```

spaceManagerDatabaseHost

`spaceManagerDatabaseHost` tells to `SRM` Space Manager which database host to connect to. Do not change unless you know what you are doing.

Default value is `localhost`.

Usage example:

```
spaceManagerDatabaseHost=database-host.example.org
```

pinManagerDatabaseHost

`pinManagerDatabaseHost` tells to `SRM` Pin Manager which database host to connect to. Do not change unless you know what you are doing.

Default value is `localhost`.

Usage example:

```
pinManagerDatabaseHost=database-host.example.org
```

srmDbName

srmDbName tells to `SRM` which database to connect to. Do not change unless you know what you are doing.

Default value is `dcache`.

Usage example:

```
srmDbName=dcache
```

srmDbUser

`srmDbUser` tells to `SRM` which database user name to use when connecting to database. Do not change unless you know what you are doing.

Default value is `srmdcache`.

Usage example:

```
srmDbUser=srmdcache
```

srmDbPassword

`srmDbPassword` tells to `SRM` which database password to use when connecting to database. Do not change unless you know what you are doing.

Usage example:

```
srmDbPassword=NotVerySecret
```

srmPasswordFile

`srmPasswordFile` tells to `SRM` which database password file to use when connecting to database. Do not change unless you know what you are doing. It is recommended that MD5 authentication method is used. To learn about file format please see http://www.postgresql.org/docs/8.1/static/libpq-pgpass.html. To learn more about authentication methods please visit http://www.postgresql.org/docs/8.1/static/encryption-options.html, Please read "Encrypting Passwords Across A Network" section.

This option is not set by default.

Usage example:

```
srmPasswordFile=/root/.pgpass
```

srmJdbsMonitoringLogEnabled

`srmJdbsMonitoringLogEnabled` tells the `SRM` to store the history of the `SRM` request executions in the database. This option is useful if you are using SRMWatch web monitoring tool. Activation of this option might lead to the increase of the database activity, so if the PostgreSQL load generated by `SRM` is excessive, disable it.

Possible values are `true` and `false`. Default is `false`.

Usage example:

```
srmJdbsMonitoringLogEnabled=false
```

srmDbLogEnabled

`srmDbLogEnabled` tells to `SRM` to store the information about the remote (copy, srmCopy) transfer details in the database. This option is useful if you are using SRMWatch web monitoring tool. Activation of this option might lead to the increase of the database activity, so if the PostgreSQL load generated by `SRM` is excessive, diable it.

Possible values are `true` and `false`. Default is `false`.

Usage example:

```
srmDbLogEnabled=false
```

srmVersion

`srmVersion` not used by SRM, it was mentioned that this value us used by some publishing scritps.

Default is `version1`.

pnfsSrmPath

`pnfsSrmPath` tells to SRM what is the root of all SRM paths is in pnfs. SRM will prepend path to all the local SURL paths passed to it by SRM client. So if the `pnfsSrmPath` is set to `/pnfs/fnal.gov/THISISTHEPNFSSRMPATH` and someone requests the read of `srm://srm.example.org:8443/file1`, SRM will translate the SURL path `/file1` into `/pnfs/fnal.gov/THISISTHEPNFSSRM-PATH/file1`. Setting this variable to something different from `/` is equivalent of performing Unix **chroot** for all SRM operations.

Default value is `/`.

Usage example:

```
pnfsSrmPath="/pnfs/fnal.gov/data/experiment"
```

parallelStreams

`parallelStreams` specifies the number of the parallel streams that SRM will use when performing third party transfers between this system and remote `GSI-FTP` servers, in response to SRM v1.1 copy or SRM V2.2 srmCopy function. This will have no effect on srmPrepareToPut and srmPrepareToGet command results and parameters of `GridFTP` transfers driven by the SRM clients.

Default value is `10`.

Usage example:

```
parallelStreams=20
```

srmBufferSize

`srmBufferSize` specifies the number of bytes to use for the in memory buffers for performing third party transfers between this system and remote `GSI-FTP` servers, in response to SRM v1.1 copy or SRM V2.2 srmCopy function. This will have no effect on srmPrepareToPut and srmPrepareToGet command results and parameters of `GridFTP` transfers driven by the SRM clients.

Default value is `1048576`.

Usage example:

```
srmBufferSize=1048576
```

srmTcpBufferSize

`srmTcpBufferSize` specifies the number of bytes to use for the tcp buffers for performing third party transfers between this system and remote `GSI-FTP` servers, in response to `SRM` v1.1 copy or `SRM` V2.2 srmCopy function. This will have no effect on srmPrepareToPut and srmPrepareToGet command results and parameters of `GridFTP` transfers driven by the `SRM` clients.

Default value is `1048576`.

Usage example:

```
srmTcpBufferSize=1048576
```

srmAuthzCacheLifetime

`srmAuthzCacheLifetime` specifies the duration that authorizations will be cached. Caching decreases the volume of messages to the `gPlazma` cell or other authorization mechanism. To turn off caching, set the value to `0`.

Default value is `120`.

Usage example:

```
srmAuthzCacheLifetime=60
```

srmGetLifeTime, srmPutLifeTime and srmCopyLife-Time

`srmGetLifeTime`, `srmPutLifeTime` and `srmCopyLifeTime` specify the lifetimes of the srmPrepareToGet (srmBringOnline) srmPrepareToPut and srmCopy requests lifetimes in millisecond. If the system is unable to fulfill the requests before the request lifetimes expire, the requests are automatically garbage collected.

Default value is `14400000` (4 hours)

Usage example:

```
srmGetLifeTime=14400000
srmPutLifeTime=14400000
srmCopyLifeTime=14400000
```

srmGetReqMaxReadyRequests, srmPutReq-MaxReadyRequests, srmGetReqReadyQueueSize and srmPutReqReadyQueueSize

`srmGetReqMaxReadyRequests` and `srmPutReqMaxReadyRequests` specify the maximum number of the files for which the transfer URLs will be be computed and given to the users in responce to `SRM`

get (srmPrepareToGet) and put (srmPrepareToPut) requests. The rest of the files that are ready to be transfered are put on the `Ready` queues, the maximum length of these queues are controlled by `srmGetReqReadyQueueSize` and `srmPutReqReadyQueueSize` parameters. These parameters should be set according to the capacity of the system, and are usually greater than the maximum number of the `GridFTP` transfers that this dCache instance `GridFTP` doors can sustain.

Usage example:

```
srmGetReqReadyQueueSize=10000
srmGetReqMaxReadyRequests=2000
srmPutReqReadyQueueSize=10000
srmPutReqMaxReadyRequests=1000
```

srmCopyReqThreadPoolSize and remoteGsiftpMaxTransfers

`srmCopyReqThreadPoolSize` and `remoteGsiftpMaxTransfers`. srmCopyReqThreadPoolSize is used to specify how many parallel srmCopy file copies to execute simultaneously. Once the this `SRM` contacted remote `SRM` system, and obtained a Transfer URL (usually `GSI-FTP` URL), it contact a Copy Manager module (usually RemoteGsiftpTransferManager), and asks it to perform a `GridFTP` transfer between remote `GridFTP` server and a dCache pool. The maximum number of the simultaneous transfers that RemoteGsiftpTransferManager will support is remoteGsiftpMaxTransfers, therefore it is important that remoteGsiftpMaxTransfers is greater than or equal to srmCopyReqThreadPoolSize.

Usage example:

```
srmCopyReqThreadPoolSize=250
remoteGsiftpMaxTransfers=260
```

srmCustomGetHostByAddr

`srmCustomGetHostByAddr` srmCustomGetHostByAddr enables using the BNL developed procedure for host by IP resolution if standard InetAddress method failed.

Usage example:

```
srmCustomGetHostByAddr=true
```

RecursiveDirectoryCreation

`RecursiveDirectoryCreation` allows or disallows automatic creation of directories via `SRM`, allow=true, disallow=false.

Automatic directory creation is allowed by default.

Usage example:

```
RecursiveDirectoryCreation=true
```

SRM Space Manager configuration

SRM Space Manager and LinkGroups

Space Manager is making reservations agains space in LinkGroups, LinkGroup is an object created by the `PoolManager`, that consists of several Links. The total space available in the given LinkGroup is a sum of available spaces in all links. The available space in each link is a sum of the available spaces in all pools assinged to a given link. Therefore for the space reservation to work correctly it is essential that each pool belongs to one and only one link, and each link belongs to only one LinkGroup. LinkGroups are assigned several parameters that determine what kind of space the LinkGroup correspond to and who can make reservation against this space.

Definition of the LinkGroups in the PoolManager.conf

To configure `PoolManager` to create the new LinkGroup (a new reservable entity in dCache), please use following example (given in the `PoolManager`). Here we assume that write-link link already exists:

```
(PoolManager) admin > psu create linkGroup write-link-group
(PoolManager) admin > psu addto linkGroup  write-link-group write-link
```

To tell Space Manager if the LinkGroup will be able to store files with given AccessLatency and RetentionPolicy, LinkGroups have 5 attributes: custodialAllowed, outputAllowed, replicaAllowed, onlineAllowed and nearlineAllowed. These attributes can be specified with the following commands:

```
(PoolManager) admin > psu set linkGroup custodialAllowed <group name> <true|false>
(PoolManager) admin > psu set linkGroup outputAllowed <group name> <true|false>
(PoolManager) admin > psu set linkGroup replicaAllowed <group name> <true|false>
(PoolManager) admin > psu set linkGroup onlineAllowed <group name> <true|false>
(PoolManager) admin > psu set linkGroup nearlineAllowed <group name> <true|false>
```

Please note that that it is up to administrators that the link groups attributes are specified correctly. For example dcache will not complain if the linkGroup that does not support tape backend will be declared as one that supports custodial.

Activating SRM Space Manager

In order to enable the new space reservation: add (uncomment) the following definition in `dCacheSetup`

```
srmSpaceManagerEnabled=yes
```

SRM Space Manager Parameters in dCacheSetup

SpaceManagerDefaultRetentionPolicy

If space reservation request does not specify retention policy we will assign `SpaceManagerDefault-RetentionPolicy` retention policy by default.

Possible values are `REPLICA`, `OUTPUT` and `CUSTODIAL`.

Usage example:

```
SpaceManagerDefaultRetentionPolicy=CUSTODIAL
```

SpaceManagerDefaultAccessLatency

If space reservation request does not specify access latency we will assign `SpaceManagerDefaultAccessLatency` this access latency by default.

Possible values are `ONLINE` and `NEARLINE`.

Usage example:

```
SpaceManagerDefaultAccessLatency=NEARLINE
```

SpaceManagerReserveSpaceForNonSRMTransfers

If `SpaceManagerReserveSpaceForNonSRMTransfers` is set to `true`, and if the transfer request come from the door, and there was not prior space reservation made for this file, Space Manager will try to reserve space before satisfying the request.

Possible values are `true` and `false`.

Usage example:

```
SpaceManagerReserveSpaceForNonSRMTransfers=false
```

SpaceManagerLinkGroupAuthorizationFileName

`SpaceManagerLinkGroupAuthorizationFileName` specifies a file that contains the list of FQANs that are allowed to make space reservations in a given link group. The file syntax is described in the next section.

This parameter is not set by default.

Usage example:

```
SpaceManagerLinkGroupAuthorizationFileName=/opt/d-cache/etc/LinkGroupAuthorization.conf
```

Implicit Space Reservations

As it was described in the section called “Introduction”, dCache can perform implicit space reservations for `SRM` Version 1 data transfers and for `SRM` Version 2.2 data transfers that are not given the space token explicitly. The parameter that enables this behavior is `srmImplicitSpaceManagerEnabled`, which is described in the section called “`SRM` configuration for experts”. In case of `SRM` version 1.1 data transfers,

when the Access Latency and Retention Policy cannot be specified, and in case of SRM V2.2 clients, when Access Latency and Retention Policy are not specified, the default values will be used. First SRM will attempt to use the values of AccessLatency and RetentionPolicy tags from the directory to which a file is being written. If the tags are present, then the Access Latency and Retention Policy will be set on basis of the system wide defaults, which are controlled by SpaceManagerDefaultRetentionPolicy and SpaceManagerDefaultAccessLatency variables in dCacheSetup; these variable are described in details in the previous section "SRM Space Manager Parameters in dCacheSetup".

If you have a direct access to the namespace, you can check if the AccessLatency and RetentionPolicy tags are present by using the following commands:

```
[root] # cd pnfsDir
[root] # cat ".(tags)()"
.(tag)(OSMTemplate)
.(tag)(file_family)
.(tag)(storage_group)
.(tag)(AccessLatency)
.(tag)(RetentionPolicy)
```

If the output contains the lines saying .(tag)(AccessLatency) and .(tag)(RetentionPolicy) than the tags are already present and you can get the actual values of these tags by executing the following commands, which are shown together with example outputs:

```
[root] # cat ".(tag)(AccessLatency)" ONLINE
[root] # cat ".(tag)(RetentionPolicy)" CUSTODIAL
```

To create/change the values of the tags, please execute :

```
[root] # echo "New Access Latency" > ".(tag)(AccessLatency)"
[root] # echo  "New Retention Policy" > ".(tag)(RetentionPolicy)"
```

The valid AccessLatency values are ONLINE and NEARLINE, valid RetentionLatency values are REPLICA, OUTPUT and CUSTODIAL.

Below are the reals examples of these commands:

```
[root] # echo "ONLINE" > ".(tag)(AccessLatency)"
[root] # echo "REPLICA" > ".(tag)(RetentionPolicy)"
```

SRM Space Manager Virtual Organization based access control configuration

VO based Authorization Prerequisites

In order to be able to take advantage of the Virtual Organization (VO) infrastructure and VO based authorization and VO Based Access Control to the Space in dCache, certain things need to be in place:

- User needs to be registered with the VO.

- User needs to use **voms-proxy-init** to create a vo proxy.
- dCache needs to use `gPlazma` and not `gPlazma` with `dcache.kpwd` plugin, but other modules that know how to extract VO attributes from the proxy. (see Chapter 12, *gPlazma authorization in dCache*, have a look at `gplazmalite-vorole-mapping` plugin.

Only if these 3 conditions are satisfied the VO based authorization of the Space Manager can work.

If a client uses a regular grid proxy, created with **grid-proxy-init**, and not a Virtual Organization (VO) proxy, which is created with the **voms-proxy-init**, when he is communicating with `SRM` server in dCache, then the VO attributes can not be extracted its credential. **voms-proxy-init** adds a Fully Qualified Attribute Name (FQAN) section(s) to the grid proxy, which contain informaton about user's VO membership, in particular it contain VO Group name and VO Role that the client intends to play at this time. In this case the name of the user is extracted on basis of the direct Distinguished Name (DN) to use name mapping. For the purposes of the space reservation the name of the user is used as its VO Group name, and the VO Role is left empty.

VO based Access Control configuration

dCache Space Reservation Functionality Access Control is currently performed at the level of the LinkGroups. The access to making reservations in each LinkGroup is controlled by the `SpaceManager-LinkGroupAuthorizationFile` property.

SpaceManagerLinkGroupAuthorizationFile syntax

The file described by `SpaceManagerLinkGroupAuthorizationFile` has following syntax:

LinkGroup Name followed by the list of the Fully Qualified Attribute Names (FQANs), each FQAN on separate line, followed by an empty line, which is used as a record separator, or by the end of file. FQAN is usually a string of the form <VO>/Role=<VORole>. Both <VO> and <VORole> could be set to `*`, in this case all VOs or VO Roles will be allowed to make reservations in this LinkGroup. Any line that starts with # is a comment and may appear anywhere.

File location is specified by defining

```
SpaceManagerLinkGroupAuthorizationFileName=FILENAME
```

in the dCacheSetup

Example of the SpaceManagerLinkGroupAuthorizationFile

```
# this is comment and is ignored

LinkGroup LFSOnly-LinkGroup
/atlas/Role=/atlas/role1

LinkGroup CMS-LinkGroup
/cms/Role=*
#/dteam/Role=/tester

LinkGroup default-LinkGroup
#allow anyone :-)
*/Role=*
```

```
#/dteam/Role=/tester
```

Successful VO and Experiment specific examples of dCache SRM Space Manager configurations are or will be published at dCache WIKI documentation pages [http://trac.dcache.org/trac.cgi/wiki/manuals/index].

SRMWatch, SRM Monitoring Tool

For large sites in order to avoid interference from Tomcat activities related to web interface, we recommend installation of SRM monitoring on a separate node.

Separate Node Installation

- Install JDK1.5
- Download, install and start latest tomcat 5.5 from Tomcat Web Site [http://tomcat.apache.org/]
- Download srmwatch RPM from http://www.dcache.org.
- Install RPM. Installation can be performed using this command:

```
[root] # rpm -Uvh srmwatch-1.0-0.i386.rpm
```

- Edit configuration file `/opt/d-cache/srmwatch-1.0/WEB-INF/web.xml` in the line saying:

```
<param-value>jdbc:postgresql://localhost/dcache</param-value>
```

  Make sure that the localhost is in jdbc url substitutes with SRM database host name. For example:

```
<param-value>jdbc:postgresql://fledgling06.fnal.gov/dcache</param-value>
```

- Execute

```
[root] # export CATALINA_HOME=YOUR_TOMCAT_LOCATION
```

- Execute

```
[root] # /opt/d-cache/srmwatch-1.0/deploy_srmwatch
```

- SRM Monitoring page should be visible at http://*srm-monitoring-node*:8080/srmwatch/

Same Node Installation

- Download srmwatch rpm from http://www.dcache.org>.
- Install rpm after srm server is installed and running, with `/opt/d-cache/etc/srm_setup.env` containing before the installation

```
TOMCAT_HTTP_ENABLED=true
```

RPM installation can be performed using this command:

```
[root] # rpm -Uvh srmwatch-1.0-0.i386.rpm
```

- SRM Monitoring page should be visible at http://*srmnode*:8080/srmwatch/

Chapter 15. dCache Web Monitoring

Vladimir Podstavkov

This part describes how to configure the web application which prepares and shows various plots using the billing information of dCache.

The monitoring system for dCache is built as a separate application using Java servlet technology and works in the Apache Tomcat servlet container. It consists from two parts - the first one works with the database where dCache system puts the detailed information about the data transfers, and the second one is a presentation layer.

Such modular architecture allows to use the tools which do best in each case. The backend part is built using Java and JDBC database driver API and is configurable using XML configuration files. The frontend part is built with OpenLaszlo technology, which allows developers to create applications with the rich user interface capabilities of desktop client software. OpenLaszlo applications are written in XML and JavaScript - just like DHTML, but portable across browsers.

The monitoring system builds the set of plots for various datasets and various time intervals - day, week, month, year... The datasets, the time intervals and data presentation are configurable via XML configuration files. The changes in the configuration can be activated remotely by the system administrator.

The total number of plots varies from system to system and currently is about 50 - 60. This system is used by public dCache, CDF, and CMS dCache systems at Fermilab.

Installation

If you want to try it for your dCache installation you can find two RPM files you need on this page: http://www.dcache.org/downloads/dcache_plots/index.shtml.

The first one (http://www.dcache.org/downloads/dcache_plots/web-dcache-*-i386.rpm) is a binary package, the second one (http://www.dcache.org/downloads/dcache_plots/web-dcache-*-src.rpm) is a source package.

To install the monitoring package do the following:

1. Install binary rpm. Installation can be performed using this command:

```
[root] # rpm -Uvh web-dcache-*-i386.rpm
```

2. Stop Tomcat if it is running

3. Untar `web-dcache.tgz` archive into Tomcat `webapps/` directory. This should be done by

```
[root] # tar xfz /opt-dcache/web-dcache/web-dcache.tgz
```

4. Install lps servlet into Tomcat `webapps/` directory

5. a. Untar `lps.tgz` archive. This should be done by

```
[root] # tar xfz /opt-dcache/web-dcache/lps.tgz
```

   b. Untar `plots.tar` archive. This should be done by

```
[root] # tar xf /opt-dcache/web-dcache/plots.tar
```

6. Copy the file `/opt/d-cache/share/java/postgresql-8.1-405.jdbc3.jar` into Tomcat `common/lib/` directory.

7. In the file `...webapps/web-dcache/META-INF/context.xml` put your real database name, database username and the password. Set its protection to 0600

```
[root] # chmod 0600 ...webapps/web-dcache/META-INF/context.xml
```

8. Start Tomcat and wait for a few minutes

9. Go to URL: http://*`your_server_name`*:*`tomcat_port_number`*/lps/plots/src/plots.lzx

Configuration

There are three XML configuration files used by the application.

1. File `.../web-dcache/WEB-INF/classes/tableConfig.xml` contains the rules for the tables creation/update. It changes only if the dCache administrator wants to add more tables into the database to work with. Here is the fragment of such file, which describes two SQL statements to create the table named en_rd_daily and to update it.

Example 15.1. Fragment of `tableConfig.xml` configuration file

```
.....................
 <table id="en_rd_daily">
   <create>
     select date(datestamp), count(*), sum(fullsize) as fullSize
     into en_rd_daily from storageinfo
     where action='restore' and errorcode=0 group by date(datestamp) order by date
   </create>
   <update>
     <query>
       delete from en_rd_daily where date between current_date-6 and current_date
     </query>
     <query>
       insert into en_rd_daily
       select date(datestamp), count(*), sum(fullsize) as fullSize
       from storageinfo
       where datestamp between current_date-6 and now() and action='restore' and errorcode=0
       group by date(datestamp) order by date
     </query>
   </update>
 </table>
.....................
```

2. File `.../web-dcache/plotConfig.xml` contains the set of commands for gnuplot program which is used to build the images. It defines the table(s) where to get data from, what data to use, and the plot format. Here is the fragment of such file, which describes plot named *billing.cst.year*, which uses the data from the table cost_daily from the billing database for the current year, and build the plot with the line and points on it.

Example 15.2. Fragment of `plotConfig.xml` configuration file

```
.....................
  <plot id="billing.cst.year">
    <title>Transaction Cost</title>
    <datasource table="cost_daily">
       date between current_date-interval '1 year' and current_date ORDER BY date
    </datasource>
    <gnusetup>
      <c>set xdata time</c>
      <c>set timefmt '%Y-%m-%d'</c>
      <c>set format x '%y-%m-%d'</c>
      <c>set xlabel 'Date (Year-month-day)'</c>
      <c>set ylabel 'Cost'</c>
      <c>###set logscale y</c>
      <dataset title="Cost" src="cost_daily">
        using {date}:{mean}  with linespoints lw 3 lt 2
       </dataset>
    </gnusetup>
  </plot>
 .....................
```

3. File `.../lps/plots/resources/pltnames.xml` contains the URLs for the generated files with the images and previews, which are shown on the web page. Here is the fragment of such file, which describes plot set named *Bytes read*, which contains the plots for a year, a month, a week, a week with one hour interval, a day, and the corresponding thumbnail previews.

Example 15.3. Fragment of `pltnames.xml` configuration file

```
.....................
  <plot>
    Bytes read
    <pic>
      <Year>http:../../../web-dcache/viewer?name=billing.brd.year.png</Year>
      <Month>http:../../../web-dcache/viewer?name=billing.brd.month.png</Month>
      <Week>http:../../../web-dcache/viewer?name=billing.brd.week.png</Week>
      <Week-daily>http:../../../web-dcache/viewer?name=billing.brd.week-daily.png</Week-daily>
      <Day>http:../../../web-dcache/viewer?name=billing.brd.day.png</Day>
    </pic>
    <pre>
      <Year>file:../../../web-dcache/viewer?name=billing.brd.year.pre</Year>
      <Month>file:../../../web-dcache/viewer?name=billing.brd.month.pre</Month>
      <Week>file:../../../web-dcache/viewer?name=billing.brd.week.pre</Week>
      <Week-daily>file:../../../web-dcache/viewer?name=billing.brd.week-daily.pre</Week-daily>
      <Day>file:../../../web-dcache/viewer?name=billing.brd.day.pre</Day>
    </pre>
  </plot>
.....................
```

Chapter 16. ACLs in dCache

Irina Kozlova

Paul Millar

Starting with the 1.9.3 series, dCache includes support for Access Control Lists (ACLs). This support is conforming to the NFS version 4 Protocol specification [http://www.nfsv4-editor.org/draft-25/draft-ietf-nfsv4-minorversion1-25.html].

This chapter provides some background information and details on configuring dCache to use ACLs and how to administer the resulting system. Some additional information can be found on the wiki page: http://trac.dcache.org/projects/dcache/wiki/Integrate.

Known limitations

At the time of writing, support for ACLs in dCache is not complete. The `SRM` component lacks support for ACLs. The other dCache doors support ACLs on all operations.

Introduction

dCache allows control over namespace operations (e.g., creating new files and directories, deleting items, renaming items) and data operations (reading data, writing data) using the standard Unix permission model. In this model, files and directories have both owner and group-owner attributes and a set of permissions that apply to the owner, permissions for users that are members of the group-owner group and permissions for other users.

Although Unix permission model is flexible enough for many deployment scenarios there are configurations that either cannot configured easily or are impossible. To satisfy these more complex permission handling dCache has support for ACL-based permission handling.

An Access Control List (ACL) is a set of rules for determining whether an end-user is allowed to undertake some specific operation. Each ACL is tied to a specific namespace entry: a file or directory. When an end-user wishes to undertake some operation then the ACL for that namespace entry is checked to see if that user is authorised. If the operation is to create a new file or directory then the ACL of the parent directory is checked.

File- and directory- ACLs

Each ACL is associated with a specific file or directory in dCache. Although the general form is the same whether the ACL is associated with a file or directory, some aspects of an ACL may change. Because of this, we introduce the terms *file-ACL* and *directory-ACL* when taking about ACLs associated with a file or a directory respectively. If the term *ACL* is used then it refers to both file-ACLs and directory-ACLs.

Each ACL contains a list of one or more Access Control Entries (ACEs). The ACEs describe how dCache determines whether an end-user is authorised. Each ACE contains information about which group of end users it applies to and describes whether this group is authorised for some subset of possible operations.

The order of the ACEs within an ACL is significant. When checking whether an end-user is authorised each ACE is checked in turn to see if it applies to the end-user and the requested operation. If it does then that ACE determines whether that end-user is authorised. If not then the next ACE is checked. Thus an ACL can have several ACEs and the first matched ACE “wins”.

One of the problems with traditional Unix-based permission model is its inflexible handling of newly created files and directories. With transitional filesystems, the permissions that are set are under the control of the user-process creating the file. The sysadmin has no direct control over the permissions that newly files or directories will have. The ACL permission model solves this problem by allowing explicit configuration using inheritance.

ACL inheritance is when a new file or directory is created with an ACL containing a set of ACEs from the parent directory’s ACL. The inherited ACEs are specially marked so that only those that are intended will be inherited.

Inheritance only happens when a new file or directory is created. After creation, the ACL of the new file or directory is completely decoupled from the parent directory’s ACL: the ACL of the parent directory may be altered without affecting the ACL of the new file or directory and visa versa.

Inheritance is optional. Within a directory’s ACL some ACEs may be inherited whilst others are not. New files or directories will receive only those ACEs that are configured; the remaining ACEs will not be copied.

ACLs and permission handlers

dCache provides support for ACLs by introducing the concept of a *permission handler*. A permission handler may be queried to determine if an end user is authorised for some file-based operation (namespace or data transfer). A permission handler will supply one of three decisions: allow, deny and defer.

dCache operates with a chain (or ordered list) of permission handlers. When determining if an end user is authorised for some operation the first permission handler in the chain is checked. If that permission handler decides that the operation should be allowed or denied for that end user then dCache will authorise or deny that user respectively. If the first permission handler defers then the next permission handler in the chain is queried.

If this second permission handler returns allow or denied the dCache will authorise or deny that user respectively. If the second permission handler also defers then dCache will continue down the chain and query the next permission handler. If the chain is exhausted then an “always deny” policy is used (TODO: is this true?).

The `UnixPermissionHandler` is an example permission handler. It provides the standard Unix permission model and decides whether an end user is authorised based on file and directory ownership and group-ownership and the set of permissions. This permission handler will always come to a decision whether an end user is authorised or not, so will never return a defer decision. Therefore any permission handler placed in the chain after a `UnixPermissionHandler` is superfluous.

The `ACLPermissionHandler` is a permission handler that implements the NFS v4 ACL model. Users and groups of users may be authorised for different operations. This is independent of a file and directory ownership. With these ACLs how new files and directories inherit permissions is described explicitly. These aspects will be described in detail later on in this chapter.

By default dCache uses the `UnixPermissionHandler`. This provides backwards compatability with older versions of dCache. To enable support for ACLs, the ACL configuration section in `dCacheSetup` file must be configured as described in the section called “permissionHandler”

Database configuration

dCache ACL support requires database tables to store ACL and ACE information. Depending on the dCache instance, this may require adjusting the database schema. This section describes the operations necessary for Chimera- and `pnfs`-based dCache instances.

Chimera

If a site has migrated from `pnfs` to Chimera then the chimera database already exists but the database schema must be extended to support ACLs. This is achieved by running two SQL files:

```
[root] # psql chimera < /opt/d-cache/libexec/chimera/sql/addACLtoChimeraDB.sql
[root] # psql chimera < /opt/d-cache/libexec/chimera/sql/pgsql-procedures.sql
```

`pnfs`

If you are not going to migrate to Chimera and you want to use ACLs with `pnfs` then you must create a database to store ACL data within. The following commands will create the database aclpnfs so it may be used to store dCache ACL data.

```
[root] # createdb aclpnfs
[root] # psql aclpnfs < /opt/d-cache/libexec/chimera/sql/create-dCacheACL.sql
```

dCache configuration

The `dCacheSetup` file contains a number of settings that may be adjusted to configure dCache’s permission settings. These settings are are described in this section.

permissionHandler

The `permissionHandler` setting describes the Permission Handler chain that dCache will use. It contains a list of permission handlers that will form the chain. Valid permission handlers are `diskCacheV111.services.acl.UnixPermissionHandler` and `diskCacheV111.services.acl.ACLPermissionHandler`. Not specifying a `permissionHandler` value is equivalent to specifying the UnixPermissionHandler.

No spaces

Remember when writing `permissionHandler` options that the value must be a single line leaving no space before and after the separating comma.

Example 16.1. Only use Unix authorisation model

The following setting results in dCache using Only the Unix authorisation model. The decision about whether an end user is authorised is based on file and directory ownership and group-ownership and on the file or directory permissions:

```
permissionHandler=diskCacheV111.services.acl.UnixPermissionHandler
```

Example 16.2. Only ACLs authorisation model

Only the NFS v4 ACL authorisation model is used:

```
permissionHandler=diskCacheV111.services.acl.ACLPermissionHandler
```

Example 16.3. ACLs first, Unix as a fall-back

The ACL permissions are checked first. If ACLs do not state whether an operation is allowed or denied then Unix permissions are checked:

```
permissionHandler=diskCacheV111.services.acl.ACLPermissionHandler,
diskCacheV111.services.acl.UnixPermissionHandler
```

Database connection

In the `ACL Configuration` section of the `dCacheSetup` file uncomment the variable `aclConnUrl` and replace the database name chimera in the URI with the name of the ACL database just created. If the database is hosted by a remote PostGreSQL instance then the host part of the URI (`localhost` in the default) must also be replaced.

If the database is aclpnfs and hosted by the PostgreSQL instance on machine `db-server.example.org`, the correct configuration line is

```
aclConnUrl=jdbc:postgresql://db-server.example.org/aclpnfs?prepareThreshold=3
```

Apply changes to all doors

The change must be applied to the `dCacheSetup` file on all doors. If the PostgreSQL instance hosting the database is on the same machine then `localhost` may be used.

Tip

Many sites can refer to a node using just the short name; for example, `db-server` instead of `db-server.example.org` in the above example.

ACL Administration

Altering dCache ACL behaviour is achieved by connecting to the `acladmin` *well-known cell* using the administrator interface. For further details about how to use the administrator interface, see the section called “The Admin Interface”.

The **info** and **help** commands are available within `acladmin` and fulfil their usual functions.

How to set ACLs

The **setfacl** command is used to set a new ACL. This command accepts arguments with the following form:

`setfacl` *`ID ACE`* [*`ACE`*...]

The *`ID`* argument is either a `pnfs`-ID or the absolute path of some file or directory in dCache. The **setfacl** command requires one or more *`ACE`* arguments seperated by spaces.

The **setfacl** command creates a new ACL for the file or directory represented by *`ID`*. This new ACL replaces any existing ACEs for *`ID`*.

An ACL has one or more ACEs. Each ACE defines permissions to access this resource for some Subject. The ACEs are space-separated and the ordering is significant. The format and description of these ACE values are described below.

Description of the ACE structure.

The *`ACE`* arguments to the **setfacl** command have a specific format. This format is described below in Extended Backus-Naur Form (EBNF).

```
[1]          ACE ::= Subject ':' Access |
                     Subject ':' Access ':' Inheritance
[2]      Subject ::= 'USER:' UserID |
                     'GROUP:' GroupID |
                     'OWNER@' |
                     'GROUP@' |
                     'EVERYONE@' |
                     'ANONYMOUS@' |
                     'AUTHENTICATED@'
[3]       Access ::= '+' Mask |
                     '-' Mask
[4]         Mask ::= Mask MaskItem |
                     MaskItem
[5]     MaskItem ::= 'r' | 'l' | 'w' | 'f' | 's' | 'a' | 'n' | 'N' | 'x' | 'd' | 'D' | 't' | 'T' | 'c' | 'C' | 'o'
[6]  Inheritance ::= Inheritance Flag |
                     Flag
[7]         Flag ::= 'f' | 'd' | 'o'
[8]       UserID ::= INTEGER
[9]      GroupID ::= INTEGER
```

The various options are described below.

The Subject

The Subject defines to which user or group of users the ACE will apply. It acts as a filter so that only those users that match the Subject will have their access rights affected.

As indicated by the EBNF above, the Subject of an ACE can take one of several forms. These are described below:

`USER:`*`id`* The `USER:` prefix indicates that the ACE applies only to the specific end-user: the dCache user with ID *`id`*. For example, `USER:0:+w` is an ACE that allows user 0 to write over a file's existing data.

`GROUP:`*`id`* The `GROUP:` prefix indicates that the ACE applies only to those end-users who are a member of the specific group: the dCache group with ID *`id`*. For example, `GROUP:20:+a` is an ACE that allows any user who is a member of group 20 to append data to the end of a file.

`OWNER@` The `OWNER@` subject indicates that the ACE applies only to whichever end-user owns the file or directory. For example, `OWNER@:+d` is an ACE that allows the file's or directory's owner to delete it.

`GROUP@` The `GROUP@` subject indicates that the ACE applies only to all users that are members of the group-owner of the file or directory. For example, `GROUP@:+l` is an ACE that allows any user that is in a directory's group-owner to list the directory's contents.

`EVERYONE@` The `EVERYONE@` subject indicates that the ACE applies to all users. For example, `EVERYONE@:+r` is an ACE that makes a file world-readable.

`ANONYMOUS@` The `ANONYMOUS@` Subject indicates that the ACE applies to all users who have not authenticated themselves. For example, `ANONYMOUS@:-l` is an ACE that prevents unauthenticated users from listing the contents of a directory.

`AUTHENTICATED@` The `AUTHENTICATED@` Subject indicates that an ACE applies to all authenticated users. For example, `AUTHENTICATED@:+r` is an ACE that allows any authenticated user to read a file's contents.

Authenticated or anonymous

An end user of dCache is either authenticated or is unauthenticated, but never both. Because of this, an end user operation will either match ACEs with `ANONYMOUS@` Subjects or `AUTHENTICATED@` Subjects but the request will never match both at the same time.

Access mask

Access (defined in the ACE EBNF above) describes what kind of operations are being described by the ACE and whether the ACE is granting permission or denying it.

An individual ACE can either grant permissions or deny them, but never both. However, an ACL may be composed of any mixture of authorising- and denying- ACEs. The first character of Access describes whether the ACE is authorising or denying.

If Access begins with a plus symbol (`+`) then the ACE authorises the Subject some operations. The ACE `EVERYONE@:+r` authorises all users to read a file since the Access begins with a `+`.

If the Access begins with a minus symbol (`-`) then the ACE denies the Subject some operations. The ACE `EVERYONE@:-r` prevents any user from reading a file since the Access begins with a `-`.

The first character of Access must be `+` or `-`, no other possibility is allowed. The initial `+` or `-` of Access is followed by one or more operation letters. These letters form the ACE's *access mask* (Mask in ACE EBNF above).

The access mask describes which operations may be allowed or denied by the ACE. Each type of operation has a corresponding letter; for example, obtaining a directory listing has a corresponding letter `l`. If a user attempts an operation of a type corresponding to a letter present in the access mask then the ACE may affect whether the operation is authorised. If the corresponding letter is absent from the access mask then the ACE will be ignored for this operation.

The following table describes the access mask letters and the corresponding operations:

File- and directory- specific operations

Some operations and, correspondingly, some access mask letters only make sense for ACLs attached to certain types of items. Some operations only apply to directories, some operations are only for files and some operations apply to both files and directories.

When configuring an ACL, if an ACE has an operation letter in the access mask that is not applicable to whatever the ACL is associated with then the letter is converted to an equivalent. For example, if `l` (list directory) is in the access mask of an ACE that is part of a file-ACL then it is converted to `r`. These mappings are described in the following table.

`r` reading data from a file. Specifying `r` in an ACE's access mask controls whether end-users are allowed to read a file's contents. If the ACE is part of a directory-ACL then the letter is converted to `l`.

`l` listing the contents of a directory. Specifying `l` in an ACE's access mask controls whether end-users are allowed to list a directory's contents. If the ACE is part of a file-ACL then the letter is converted to `r`.

`w` overwriting a file's existing contents. Specifying `w` in an ACE's access mask controls whether end-users are allowed to write data anywhere within the file's current offset range. This includes the ability to write to any arbitrary offset and, as a result, to grow the file. If the ACE is part of a directory-ACL then the letter is converted to `f`.

`f` creating a new file within a directory. Specifying `f` in an ACE's access mask controls whether end-users are allowed to create a new file. If the ACE is part of an file-ACL then then the letter is converted to `w`.

`s` creating a subdirectory within a directory. Specifying `s` in an ACE's access mask controls whether end-users are allowed to create new subdirectories. If the ACE is part of a file-ACL then the letter is converted to `a`.

`a` appending data to the end of a file. Specifying `a` in an ACE's access mask controls whether end-users are allowed to add data to the end of a file. If the ACE is part of a directory-ACL then the letter is converted to `s`.

`n` reading attributes. Specifying `n` in an ACE's access mask controls whether end-users are allowed to read attributes. This letter may be specified in ACEs that are part of a file-ACL and those that are part of a directory-ACL.

`N` write attributes. Specifying `N` in an ACE's access mask controls whether end-users are allowed to write attributes. This letter may be specified in ACEs that are part of a file-ACL and those that are part of a directory-ACL.

`x` executing a file or entering a directory. `x` may be specified in an ACE that is part of a file-ACL or a directory-ACL; however, the operation that is authorised will be different.

Specifying `x` in an ACEs access mask that is part of a file-ACL will control whether end users matching the ACE Subject are allowed to execute that file.

Specifying `x` in an ACEs access mask that is part of a directory-ACL will control whether end users matching ACE Subject are allowed to search a directory for a named file or subdirectory. This operation is needed for end users to change their current working directory.

`d` deleting a namespace entry. Specifying `d` in an ACE's access mask controls whether end-users are allowed to delete the file or directory the ACL is attached. The end user must be also authorised for the parent directory (see `D`).

`D` deleting a child of a directory. Specifying `D` in the access mask of an ACE that is part of a directory-ACL controls whether end-users are allowed to delete items within that directory. The end user must be also authorised for the existing item (see `d`).

`t` reading basic attributes. Specifying `t` in the access mask of an ACE controls whether end users are allowed to read basic (i.e., non-ACL) attributes of that item.

`T` altering basic attributes. Specifying `T` in an ACE's access mask controls whether end users are allowed to alter timestamps of the item the ACE's ACL is attached.

`c` reading ACL information. Specifying `c` in an ACE's access mask controls whether end users are allowed to read the ACL information of the item to which the ACE's ACL is attached.

`C` writing ACL information. Specifying `C` in an ACE's access mask controls whether end users are allowed to update ACL information of the item to which the ACE's ACL is attached.

`o` altering owner and owner-group information. Specifying `o` controls whether end users are allowed to change ownership information of the item to which the ACE's ACL is attached.

ACL inheritance

To enable ACL inheritance, the optional inheritance flags must be defined. The flag is a list of letters. There are three possible letters that may be included and the order doesn't matter.

Chimera only

Note that inheritance is only available with Chimera. When using ACLs with `pnfs`, there is no inheritance of ACEs on the newly created objects (files or directories).

To provide a similar functionality for a `pnfs`-based dCache instance, the system administrator will have to set ACLs on the newly created objects external to dCache, either manually or by writing a script.

ACE Inheritance Flags

`f` This inheritance flag only affects those ACEs that form part of an directory-ACL. If the ACE is part of a file-ACL then specifying `f` has no effect.

If a file is created in a directory with an ACE with `f` in inheritance flags then the ACE is copied to the newly created file's ACL. This ACE copy will not have the `f` inheritance flag.

Specifying `f` in an ACE's inheritance flags does not affect whether this ACE is inherited by a newly created subdirectory. See `d` for more details.

`d` This inheritance flag only affect those ACEs that form part of an directory-ACL. If the ACE is part of a file-ACL then specifying `d` has no effect.

Specifying `d` in an ACE's inheritance flags does not affect whether this ACE is inherited by a newly created file. See `f` for more details.

If a subdirectory is created in a directory with an ACE with `d` in the ACE's inheritance flag then the ACE is copied to the newly created subdirectory's ACL. This ACE copy will have the `d` inheritance flag specified. If the `f` inheritance flag is specified then this, too, will be copied.

`o` The `o` flag may only be used when the ACE also has the `f`, `d` or both `f` and `d` inheritance flags.

Specifying `o` in the inheritance flag will suppress the ACE. No user operations will be authorised or denied as a result of such an ACE.

When a file or directory inherits from an ACE with `o` in the inheritance flags then the `o` is *not* present in the newly created file or directory's ACE. Since the newly created file or directory will not have the `o` in it's inheritance flags the ACE will take effect.

An `o` in the inheritance flag allows child files or directories to inherit authorisation behaviour that is different from the parent directory.

Examples

This section gives some specific examples of how to set ACLs to achieve some specific behaviour.

Example 16.4. ACL allowing specific user to delete files in a directory

This example demonstrates how to configure a directory-ACL so user 3750 can delete any file within the directory `/pnfs/example.org/data/exampleDir`.

```
(acladmin) admin > setfacl /pnfs/example.org/data/exampleDir EVERYONE@:+l USER:3750:D
    (...line continues...)   USER:3750:+d:of
(acladmin) admin > setfacl /pnfs/example.org/data/exampleDir/existingFile1
    (...line continues...)   USER:3750:+d:f
(acladmin) admin > setfacl /pnfs/example.org/data/exampleDir/existingFile2
    (...line continues...)   USER:3750:+d:f
```

The first command creates an ACL for the directory. This ACL has three ACEs. The first ACE allows anyone to list the contents of the directory. The second ACE allows user 3750 to delete content within the directory in general. The third ACE is inherited by all newly created files and specifies that user 3750 is authorised to delete the file independent of that file's ownership.

The second and third commands creates an ACL for files that already exists within the directory. Since ACL inheritance only applies to newly created files or directories, any existing files must have an ACL explicitly set.

Example 16.5. ACL to deny a group

The following example demonstrates authorising all end users to list a directory. Members of group 1000 can also create subdirectories. However, any member of group 2000 can do neither.

```
(acladmin) admin > setfacl /pnfs/example.org/data/exampleDir GROUP:2000:-sl
    (...line continues...)     EVERYONE@:+l GROUP:1000:+s
```

The first ACE denies any member of group 2000 the ability to create subdirectories or list the directory contents. As this ACE is first, it takes precedence over other ACEs.

The second ACE allows everyone to list the directory's content. If an end user who is a member of group 2000 attempts to list a directory then their request will match the first ACE so will be denied. End users attempting to list a directory that are not a member of group 2000 will not match the first ACE but will match the second ACE and will be authorised.

The final ACE authorises members of group 1000 to create subdirectories. If an end user who is a member of group 1000 and group 2000 attempts to create a subdirectory then their request will match the first ACE and be denied.

Example 16.6. ACL to allow a user to delete all files and subdirectories

This example is an extension to Example 16.4, "ACL allowing specific user to delete files in a directory". The previous example allowed deletion of the contents of a directory but not the contents of any subdirectories. This example allows user 3750 to delete all files and subdirectories within the directory.

```
(acladmin) admin > setfacl /pnfs/example.org/data/exampleDir USER:3750:+D:d
    (...line continues...)     USER:3750:+d:odf
```

The first ACE is `USER:3750:+D:d`. This authorises user 3750 to delete any contents of directory `/pnfs/example.org/data/exampleDir` that has an ACL authorising them with `d` operation.

The first ACE also contains the inheritance flag `d` so newly created subdirectories will inherit this ACE. Since the inherited ACE will also contain the `d` inheritance flag, this ACE will be copied to all subdirectories when they are created.

The second ACE is `USER:3750:+d:odf`. The ACE authorises user 3750 to delete whichever item the ACL containing this ACE is associated with. However, since the ACE contains the `o` in the inheritance flags, user 3750 is *not* authorised to delete the directory `/pnfs/example.org/data/exampleDir`

Since the second ACE has both the `d` and `f` inheritance flags, it will be inherited by all files and subdirectories of `/pnfs/example.org/data/exampleDir`, but without the `o` flag. This authorises user 3750 to delete these items.

Subdirectories (and files) will inherit the second ACE with both `d` and `f` inheritance flags. This implies that all files and sub-subdirecties within a subdirectory of `/pnfs/example.org/data/exampleDir` will also inherit this ACE, so will also be deletable by user 3750.

Viewing configured ACLs

The **getfacl** is used to obtain the current ACL for some item in dCache namespace. It takes the following arguments.

`getfacl` [*`pnfsId`*] | [*`globalPath`*]

The **getfacl** command fetches the ACL information of a namespace item (a file or directory). The item may be specified by its `pnfs`-ID or its absolute path.

Example 16.7. Obtain ACL information by absolute path

```
(acladmin) admin > getfacl /pnfs/example.org/data/exampleDir
ACL: rsId = 00004EEFE7E59A3441198E7EB744B0D8BA54, rsType = DIR
order = 0, type = A, accessMsk = lfsD, who = USER, whoID = 12457
order = 1, type = A, flags = f, accessMsk = lfd, who = USER, whoID = 87552
In extra format:
USER:12457:+lfsD
USER:87552:+lfd:f
```

The information is provided twice. The first part gives detailed information about the ACL. The second part, after the `In extra format:` heading, provides a list of ACEs that may be used when updating the ACL using the **setfacl** command.

Chapter 17. GLUE info provider

This chapter describes how to configure the GLUE information provider supplied with dCache so it provides the correct information. This is necessary so that WLCG infrastructure (such as FTS) and clients using WLCG tools can correctly use the dCache instance.

The process is designed to be the minimum overhead so it can easily be performed manually; however, you may choose to use an automatic configuration tool, such as YAIM.

Warning

Please be aware that changing information provider may result in a brief interruption to published information. This may have an adverse affect on client software that make use of this information.

Ensuring dCache information is available

Make sure that both the `httpd` and `info` services are running; both are required for publishing information. By default, the `info` service is started as an admin-node responsibility; but it is possible to configure dCache so it runs on a different node. You should run only one info service per dCache instance.

The `info` service is not the `infoProvider` service

The info-provider needs accurate, up-to-date information about a dCache instance so it can publish correct values. A component of dCache, rather confusingly also called `infoProvider`, used to provide this up-to-date information. By default, the `infoProvider` would run in its own domain (`infoProviderDomain`) and could be started and stopped like any other domain.

The job of collecting accurate, up-to-date information is now handled by the `info` service. This is completely independent of the `infoProvider` service, so the latter is no longer needed and may be switched off.

For more details please see the section called “Decommissioning the old info provider”.

If necessary, you may start the info service manually:

```
[root] # /opt/d-cache/bin/dcache start infoDomain
Starting infoDomain done
```

You can check which services are running on the local node using the status command:

```
[root] # /opt/d-cache/bin/dcache status
Domain                    Status       PID
dCacheDomain              running    30582
dirDomain                 running    30625
adminDoorDomain           running    30667
httpdDomain               running    30711
utilityDomain             running    30760
gPlazma-dcache-hostDomain running    30844
namespaceDomain           running    30921
dcache-hostDomain         running    30971
infoDomain                running    15530
```

Output may look different

One feature of dCache is that domains may be run on different nodes. Because of this, the list of domains running on the node running the `info` service may be different.

You can also verify both services (`httpd` and `info`) are running with the following **wget** command. This command assumes that you run it on the node that has the `httpd` service (by default, the admin node). If may run the command on any node by replacing `localhost` with the hostname of the node running the `httpd` service.

The following example shows the output when the `info` service is running correctly

```
[root] # wget -O/dev/null http://localhost:2288/info
--17:57:38--  http://localhost:2288/info
Resolving localhost... 127.0.0.1
Connecting to localhost|127.0.0.1|:2288... connected.
HTTP request sent, awaiting response... 200 Document follows
Length: 372962 (364K) [application/xml]
Saving to: `/dev/null'

100%[======================================================================
===>] 372,962     --.-K/s   in 0.001s

17:57:38 (346 MB/s) - `/dev/null' saved [372962/372962]
```

If the `httpd` service isn't running then the command will generate the following output:

```
[root] # wget -O/dev/null http://localhost:2288/info
  --10:05:35--  http://localhost:2288/info
             => `/dev/null'
  Resolving localhost... 127.0.0.1
  Connecting to localhost|127.0.0.1|:2288... failed: Connection refused.
```

To fix the problem, start the `httpd` service with the following command on the appropriate dCache node.

```
[root] # /opt/d-cache/bin/dcache start httpdDomain
Starting httpdDomain done
```

If running the **wget** command gives the following output:

```
[root] # wget -O/dev/null http://localhost:2288/info
  --10:03:13--  http://localhost:2288/info
             => `/dev/null'
  Resolving localhost... 127.0.0.1
  Connecting to localhost|127.0.0.1|:2288... connected.
  HTTP request sent, awaiting response... 503 Unable to contact the info cell.  Pl
ease ensure the info cell is running.
  10:03:13 ERROR 503: Unable to contact the info cell.  Please ensure the info cel
l is running..
```

then the `info` service is not running. Instructions for starting the `info` service are given above.

Configuring the info provider

In the directory `/opt/d-cache/etc` you will find a template file `glue-1.3.xml.template`. Copy this file as `glue-1.3.xml` in the same directory.

Edit `glue-1.3.xml` with your favourite text editor.

The file is split into two parts. The first part contains the configuration that a site will typically need to configure. Most sites may leave the second part alone. The two parts are separated by the comment:

```
  <!--+
      |    YOU SHOULD NOT NEED TO EDIT ANYTHING BELOW THIS POINT.
      +-->
```

Take care when editing the file! After changing the contents, the file must remain valid, well-formed XML. In particular, be very careful not to add a less-than symbol (`<`) that isn't part of an XML element, or an ampersand symbol (`&`) that isn't part of an entity markup. If you wish to either symbol you must use the marked-up version: `<` and `&` respectively. For example:

```
<constant id="SE-NAME">Simple &amp; small dCache instance for small VOs
(typically &lt; 20 users)</constant>
```

As a further hint, you should *only* edit text between two elements or add more elements (for lists and mappings). You should *never* alter the text inside double-quote marks. For example, with the following element definition:

```
<constant id="SITE-UNIQUE-ID">EXAMPLESITE-ID</constant>
```

you should review the contents between the elements (`EXAMPLESITE-ID`) and edit the contents accordingly. You should *never* edit the `SITE-UNIQUE-ID` as it is in double-quote marks. A valid, edited value is:

```
<constant id="SITE-UNIQUE-ID">DESY-HH</constant>
```

Testing the info provider

Once you have configured `glue-1.3.xml` to reflect your site's configuration, you may test that the info provider produces meaningful results.

Run the info-provider script should produce GLUE information in LDIF format; for example:

```
[root] # /opt/d-cache/libexec/infoProvider/info-based-infoProvider.sh
  #
  #  LDIF generated by Xylophone v0.1
  #
  #  XSLT processing using libxslt 1.0 (http://xmlsoft.org/XSLT/)
  #


  dn: GlueSEUniqueID=dcache-host.example.org,mds-vo-name=resource,o=grid
  objectClass: GlueSETop
  objectClass: GlueSE
  objectClass: GlueKey
  objectClass: GlueSchemaVersion
  GlueSEStatus: Production
  GlueSEUniqueID: dcache-host.example.org
  GlueSEArchitecture: multidisk
  GlueSEImplementationName: dcache
Many similar lines follow
```

The actual values you see will be site-specific and depend on the contents of the `glue-1.3.xml` file and your dCache configuration.

To verify that there are no problems, redirect standard-out to `/dev/null` to show only the error messages:

```
[root] # /opt/d-cache/libexec/infoProvider/info-based-infoProvider.sh >/dev/null
```

If you see two error messages, which may be repeated several times, of the form:

```
[root] # /opt/d-cache/libexec/infoProvider/info-based-infoProvider.sh >/dev/null
  error : Operation in progress
  warning: failed to load external entity "http://localhost:2288/info"
```

then it is likely that either the `httpd` or `info` service have not been started. Use the above **wget** test to check that both services are running. You can also see which services are available by running the command: **`dcache status`**.

Decommissioning the old info provider

Sites that were previously using the old (Java-based) info provider should ensure that they no longer using it. This is because, if so configured, GIP will obtain information from both the (new) info-based info provider and the Java-based info provider and attempt to merge the results. This will likely lead to a confusing description of dCache, which may prevent clients from working correctly.

The Java-based info provider has two configuration files and a symbolic link within GIP. They are:

- The file `/opt/lcg/var/gip/ldif/lcg-info-static-SE.ldif`,
- The file: `/opt/lcg/var/gip/ldif/lcg-info-static-dSE.ldif`,
- The symbolic link `/opt/glite/etc/gip/plugin`, which points to `/opt/d-cache/jobs/infoDynamicSE-plugin-dcache`.

The two files (`lcg-info-static-SE.ldif` and `lcg-info-static-dSE.ldif`) may appear within a different directory if the `static_dir` variable is configured. You will find the `static_dir` variable in one of two configuration files: either `/opt/glite/etc/gip/glite-info-generic.conf` or `/opt/lcg/etc/lcg-info-generic.conf`.

Delete the above three entries: `lcg-info-static-SE.ldif`, `lcg-info-static-dSE.ldif` and the `plugin` symbolic link.

The directory defined in the `static_dir` variable (`/opt/lcg/var/gip/ldif` by default) may contain other static LDIF entries that are relics of previous info-providers. These may have filenames like `static-file-SE.ldif`.

Delete any such files that contain information about dCache. All LDIF information now comes directly from the info-provider and there should be no static LDIF files.

The `infoProvider` component of dCache, usually running in its own domain (`infoProviderDomain`), collects information for the infoDynamicSE-plugin-dcache program. Since this program is now obsolete the `infoProviderDomain` should be switched off. This may be achieved with the **dcache** script:

```
[root] # /opt/d-cache/bin/dcache stop infoProviderDomain
Stopping infoProviderDomain (pid=15528) 0 done
```

You can confirm that the `infoProviderDomain` domain is no longer running by querying the current status of the components:

```
[root] # /opt/d-cache/bin/dcache status | grep ^infoProvider
  infoProviderDomain      stopped         /var/log/infoProviderDomain.log
```

To prevent dCache from starting the `infoProviderDomain` domain when the machine next reboots, you should edit `/opt/d-cache/etc/node_config` and ensure that the `infoProvider` option is configured to `no`. You can verify that `infoProviderDomain` is no longer listed as a service with the following command:

```
[root] # /opt/d-cache/bin/dcache services | grep ^infoProvider
```

You should see no output from running the command.

Publishing information from the info-provider

By default BDII obtains fresh information by querying GIP. To allow BDII to obtain the GLUE information, you must tell GIP where to find this information. This is achieved by either copying the above script into the directory `/opt/glite/etc/gip/provider/`

```
[root] # cp /opt/d-cache/libexec/infoProvider/info-based-infoProvider.sh \
/opt/glite/etc/gip/provider/
```

or by symbolically linking the script in there:

```
[root] # ln -s /opt/d-cache/libexec/infoProvider/info-based-infoProvider.sh
/opt/glite/etc/gip/provider/
```

If GIP (available in the `glite-info-generic` RPM package) and BDII are installed, and the BDII daemons are running, then you will see the information appear in BDII after a short delay; by default, this is 60 seconds.

You can verify the information is present with the query:

```
[root] # ldapsearch -LLL -x -H ldap://dcache-host:2170 \
-b o=grid
dn: o=grid
objectClass: organization
o: grid

dn: Mds-Vo-name=local,o=grid
objectClass: Mds
Mds-Vo-name: local

dn: Mds-Vo-name=resource,o=grid
```

```
objectClass: Mds
Mds-Vo-name: resource

dn: GlueSEUniqueID=dcache-host.example.org,Mds-Vo-name=resource,o=grid
GlueSEStatus: Production
objectClass: GlueSETop
objectClass: GlueSE
objectClass: GlueKey
objectClass: GlueSchemaVersion
GlueSETotalNearlineSize: 2000
GlueSEArchitecture: tape
GlueSEName: SRM-DEVEL
GlueSchemaVersionMinor: 3
GlueSEUsedNearlineSize: 0
GlueChunkKey: GlueSEUniqueID=dcache-host.example.org
GlueForeignKey: GlueSiteUniqueID=EXAMPLE-SITE
GlueSchemaVersionMajor: 1
GlueSEImplementationName: dCache
GlueSEUniqueID: dcache-host.example.org
GlueSETotalOnlineSize: 4832
GlueSESizeTotal: 4832
GlueSESizeFree: 4832
GlueSEUsedOnlineSize: 0
GlueSEImplementationVersion: 1.9.5-16 (ns=Chimera)

dn: GlueSALocalID=tape-atlas,GlueSEUniqueID=dcache-host.example.org,Mds-Vo-name=reso
 urce,o=grid
GlueSATotalNearlineSize: 1000
objectClass: GlueSATop
objectClass: GlueSA
objectClass: GlueSAAccessControlBase
objectClass: GlueSAState
objectClass: GlueSchemaVersion
objectClass: GlueKey
GlueSAExpirationMode: neverExpire
GlueChunkKey: GlueSEUniqueID=dcache-host.example.org
GlueSAReservedOnlineSize: 0
GlueSACapability: InstalledOnlineCapacity=0
GlueSACapability: InstalledNearlineCapacity=1000
GlueSchemaVersionMinor: 3
GlueSAUsedNearlineSize: 0
GlueSAName: Tapes for ATLAS
GlueSAUsedOnlineSize: 0
GlueSAFreeOnlineSize: 0
GlueSAFreeNearlineSize: 1000
GlueSAReservedNearlineSize: 1000
GlueSchemaVersionMajor: 1
GlueSAAccessControlBaseRule: VO:atlas
GlueSALocalID: tape-atlas
```

There are likely many further objects defined. These objects have been omitted for brevity.

Don't use `localhost`

It's tempting to use `localhost` as the hostname in the **ldapsearch** command above. Unfortunately, for some versions of BDII this will not work. Recent versions of BDII bind to the ethernet device (e.g., eth0). Typically, `localhost` is associated with the loopback device (lo), so the LDAP server part of BDII will not hear the connection request and the query will fail.

You should be able to compare this output with the output from running the info-provider script manually. If the info-provider output includes LDAP objects that are absent in the BDII output then there is a problem somewhere. The BDII log file will likely explain why this object was not accepted; for example, due to a badly formatted attribute.

Unfortunately, the order of the LDAP objects and the order of the attributes within the object (other than the distinguished name, DN, which must be first) are not required to be in any particular order. Because of this one cannot use the **diff** command to look for changes.

Updating information

The information contained within the `info` service may take a short time to achieve a complete overview of dCache's state. For certain gathered information it may take a few minutes before the information stabilises. This delay is intentional and prevents the gathering of information from adversely affecting dCache's performance.

The information presented by the LDAP server is updated, by BDII, periodically by requesting fresh information from the info-provider. The info-provider obtains this information by requesting dCache's current status from `info` service. By default, BDII will query the info-provider every 60 seconds. This will introduce an additional delay between a change in dCache's state and that information propagating.

A few pieces of information are hard-coded within the `glue-1.3.xml` file; that is, you will need to edit this file before the published value(s) will change. These values are ones that typically a site-admin must choose independently of dCache's current operations.

Troubleshooting BDII problems

The BDII log files may show entries like:

```
==> slapadd: could not parse entry (line=26)
Error nearby dn:
GlueVOInfoLocalID=atl[…],o=grid ==>
str2entry: invalid value for attributeType GlueSATotalOnlineSize #0
(syntax 1.3.6.1.4.1.1466.115.121.1.27)
```

This kind of problem comes when BDII is attempting to inject new information into the OpenLDAP server. This server is rejecting some of that information because it is badly formatted. In this example, `1.3.6.1.4.1.1466.115.121.1.27` is LDAP-speak for "an integer number". The offending attribute has a value of `#0`, which isn't an integer number as it starts with the hash symbol (#).

The `nearby dn` description of the report should be ignored. It is unclear how OpenLDAP decides which object is "nearby", but it's usually inaccurate. Instead, the important piece of information is the line number (`line=26` in above). BDII injects fresh information into the OpenLDAP server from a file and the line number mentioned in the error message is from this file.

The following two sections describe how to locate the injection file for BDII v4 and v5. In those sections, when mentioning file locations, the default location has been included in parentheses. Since many sites deploy BDII with minimal changes the default locations are likely correct.

Locating BDII v4 injection LDIF files

The default location is `/opt/bdii/var/cache/1/GIP.ldif`

The file `/opt/bdii/etc/bdii.conf` contains various configuration options for BDII; for example, if this file has the line `BDII_DIR=/opt/bdii` then `BDII_DIR` will take the value `/opt/bdii`.

The file *BDII_DIR*/etc/bdii-update.conf (/opt/bdii/etc/bdii-update.conf) contains a list describing methods for obtaining LDIF information. When updating the contents of a server, BDII will obtain LDIF information from each source and attempt to inject the data into the OpenLDAP server. The bdii-update.conf file will likely contain a single line:

```
GIP file:///opt/glite/libexec/glite-info-wrapper
```

Each line in bdii-update.conf has two parts. The second part is a URI describing how to obtain the data. If the URI starts file:// (as in the above example) then the file is executed and the LDIF is taken from the standard-output of the process. The first part (GIP in above example) is used to specify the filename for the combined output (GIP.ldif) within the *BDII_VAR_DIR*/cache directory (/opt/bdii/var/cache) hierarchy.

The output from the various sources (as defined in the bdii-update.conf file) are stored as files within the *BDII_VAR_DIR*/cache/0 directory (/opt/bdii/var/cache/0). After fresh data is injected, the cache directories are advanced one place, so the directory cache/0 becomes cache/1, directory cache/1 becomes cache/2, and so on. The result is that, after injecting, the LDIF output will be in directory *BDII_VAR_DIR*/cache/1 (/opt/bdii/var/cache/1) and will be contained within the file GIP.ldif.

Locating BDII v5 injection LDIF files

The default location is /var/bdii/old.ldif

As with BDII v4, the file /opt/bdii/etc/bdii.conf contains configuration, such as in which directories information will be stored.

The update process in BDII v5 is different from v4, so the generated files are different. With v5, BDII maintains knowledge of the current state of the LDAP server in the file BDII_VAR_DIR/old.ldif (/var/bdii/old.ldif). It uses this information to generate a description of how to modify the local BDII server (by adding, removing and modifying attributes and objects as necessary).

Once the injection LDIF file is located, the line number may be used to discover which object is causing the problem, so enabling further investigation.

Chapter 18. Stage Protection

Irina Kozlova

Initially dCache has been designed to be a disk cache in front of a Tape Storage System, moving files onto the tape-backend and restoring them when needed. Those operations are handled transparently to the user. The downside of this approach is that a simple read of a file, not being on disk, automatically triggers a tape operation. As tape operations are expensive and may interfere with storing raw data, coming from the Tier 0, this feature had to be reviewed. As a result, it has been agreed with the experiments that only a production user should be allowed to trigger such a tape operation. dCache is now implementing a first version of such a protective mechanism.

A dCache system administrator may specify a list of DNs/FQANs which are allowed to trigger tape read accesses for files not being available on disk. Users, requesting tape-only files, and not being on that *white list*, will receive a permission error and no tape operation is launched. In 1.9.5-20 release, stage protection can be enhanced to allow authorization specific to a dCache storage group. The additional configuration parameter is optional allowing the stage protection to be backwards compatible when stage authorization is not specific to a storage group.

Configuration of Stage Protection

In the 1.9.4 series, stage protection had to be configured in every door and in the `PinManager`. Starting with the 1.9.5 series, stage protection can optionally be configured in the `PoolManager` rather than on the doors and `PinManager`. Thus the white list needs to be present on a single node only. To enable this, define the following parameter in `config/dCacheSetup`:

```
stagePolicyEnforcementPoint=PoolManager
```

The file name of the white list must be configured by setting the `stageConfigurationFilePath` parameter in `config/dCacheSetup`:

```
stageConfigurationFilePath=${ourHomeDir}/config/StageConfiguration.conf
```

The parameter only needs to be defined on the nodes which enforce the stage protection; i.e., either on the doors and `PinManager`, or in `PoolManager`.

Definition of the White List

The Stage Configuration File will contain a white list. Each line of the white list may contain one or two regular expressions enclosed in double quotes. The first regular expression matches the DN, and the second matches the FQAN :

"*DN*" ["*FQAN*"]

Lines starting with a hash symbol # are discarded as comments.

The regular expression syntax follows the syntax defined for the Java Pattern class [http://java.sun.com/javase/6/docs/api/java/util/regex/Pattern.html].

Here are some examples of the White List Records for the 1.9.5 series:

```
".*" "/atlas/Role=production"
"/C=DE/O=DESY/CN=Kermit the frog"
"/C=DE/O=DESY/CN=Beaker" "/desy"
"/O=GermanGrid/.*" "/desy/Role=.*"
```

This example authorizes a number of different groups of users:

- Any user with the FQAN `/atlas/Role=production`.
- The user with the DN `/C=DE/O=DESY/CN=Kermit the frog`, irrespective of which VOMS groups he belongs to.
- The user with the DN `/C=DE/O=DESY/CN=Beaker` but only if he is also identified as a member of VO `desy` (FQAN `/desy`)
- Any user with DN and FQAN that match `/O=GermanGrid/.*` and `/atlas/Role=.*` respectively.

With the plain `dCap` protocol the DN and FQAN are not known for any users. Therefore, there is a special case for `dCap` users in 1.9.5. In order to allow all `dCap` users to stage files the white list should contain the following record:

```
"Unknown" "nobody"
```

If this line is commented or not present in the white list, all `dCap` users will be disallowed to stage files.

Authorizing Only Certain Storage Groups

In 1.9.5-20 release, an optional storage group parameter can be specified. Each line of the white list may contain up to three regular expressions enclosed in double quotes. The regular expressions match the DN, FQAN, and Storage Group written in the following format:

"*DN*" ["*FQAN*" ["*StorageGroup*"]]

If a storage group is specified all three parameters must be provided. The regular expression `".*"` may be used to authorize any DN or any FQAN. Consider the following example:

```
".*" "/atlas/Role=production" "h1:raw@osm"
"/C=DE/O=DESY/CN=Scooter" ".*" "sql:chimera@osm"
```

In the example above:

- Any user with FQAN `/atlas/Role=production` is allowed to stage files located in the storage group `h1:raw@osm`.
- The user `/C=DE/O=DESY/CN=Scooter`, irrespective of which VOMS groups he belongs to, is allowed to stage files located in the storage group `sql:chimera@osm`.

In the following example, all `dCap` users are allowed to stage files located in the storage group `h1:raw@osm`:

```
"Unknown" "nobody" "h1:raw@osm"
```

Part III. Cookbook

Table of Contents

This part contains guides for specific tasks a system administrator might want to perform.

Chapter 19. General

Installing dCache on Opteron Machines

The PNFS server, dCache server, and dCache client software have to be taken care of:

The dCache Server

The major part of the dCache server software is written in Java. Therefore the Java Virtual Machine with 64 bit extension needs to be installed. It either is included in the regular Java distribution or additional packages have to be downloaded.

The dCache Client

The *dCap library* and the command line tool **dccp** may be downloaded from http://www.dcache.org/downloads/ [http://www.dcache.org/downloads.shtml] for several architectures. The source of the client software may also be downloaded from http://www.dcache.org/downloads/cvs.shtml [http://www.dcache.org/downloads/cvs.shtml] and compiled. As of this writing, this has not been tested for the Opteron architecture. Please, contact <support@dcache.org> when encountering any problems with this.

pnfs Server

The current version of the pnfs server software is written in C and has never been compiled for any 64-bit architecture. Since a Java implementation is in preparation, there are no plans to do that. Therefore the pnfs server has to be run in “compat mode”.

Chapter 20. Pool Operations

Enabling checksums

How to enable checksums

The following section describes how to enable checksum calculation on write transfers with maximum security. Two checksums will be computed on different points for each transfer: on the fly during file arrival and once the file was completely written to disk . Both checksums are compared with each other and (if available) with another checksum sent by the client. If, and only if, all of them match, the transfer is considered to be successful and the checksum is stored in `pnfs`.

To enable checksumming (independent from access protocol type), make sure the following option appears in the `pool.batch`-file:

```
define context startPools endDefine
  create diskCacheV111.pools.MultiProtocolPool2 ${0} \
  ..
  -calculate-transfer-crc \
  ..
"
```

Additionally, the checksum policy must be customized accordingly. This is done by modifying the pool-setup-file (found at *`poolPath`*`/pool/setup`) such that it contains the following line:

```
csm set policy -onwrite=on -ontransfer=on -enforcecrc=on
```

Now a restart of the pool should activate all changes. Please repeat the upper procedure on all write-pools you want to have checksum-enabled.

Warning

Please note that the following policy options should *not* be touched:

getcrcfromhsm	this option is tailored to DESY's HSM and won't work anywhere else
onread	reserved for future use, no checksum handling on read transfers for now.
frequently	reserved for future use (recalculating checksums for files residing in the pool on a regular basis).

The default pool behavior

When setting up a pool from scratch, the default policy is to calculate only the checksum on the file written to disk, but not on the fly upon arrival. In case there is a client checksum available (always true for `dCap`), they get compared and must match. Otherwise, the checksum computed on the written disk file will be stored in `pnfs` instead.

To reset the default behavior, set the following line in the pool-setup-file and restart the pool:

```
csm set policy -onwrite=on -enforcecrc=on
```

Checksums in detail

Overview

When writing data into the dCache, and possibly later on into an HSM, checksums may be calculated at different points within this chain.

Client Checksum
: The client calculates the checksum before or while the data is sent to the dCache. The checksum value, depending on when it has been calculated, may sent together with the open request to the door and stored into `pnfs` before the data transfer begins or it may be sent with the close operation after the data has been transferred.

  The `dCap` protocol providing both methods, but the `dCap` clients use the latter by default.

  The `FTP` protocol does not provide a mechanism to send a checksum. Nevertheless, some `FTP` clients can (mis-)use the "`site`" command to send the checksum prior to the actual data transfer.

Transfer Checksum
: While data is coming in, the server data mover may calculate the checksum on the fly.

Server File Checksum
: After all the file data has been received by the dCache server and the file has been fully written to disk, the server may calculate the checksum, based on the disk file.

The graph below sketches the different schemes for `dCap` and `FTP` with and without client checksum calculation:

Table 20.1. Checksum calculation flow

Step	FTP (w/o initial CRC)	FTP (with initial CRC)	dCap
1	Create Entry		
2		Store Client CRC in `pnfs`	
3	Server calculates transfer CRC		
4		Get Client CRC from `pnfs`	Get Client CRC from mover
5		Compare Client and Server CRC	
6	Store transfer CRC in `pnfs`		Store client CRC in `pnfs`
7	Server calculates disk file CRC		

ChecksumMover Interface

As far as the server data mover is concerned, only the *Client Checksum* and the *Transfer Checksum* are of interrest. While the client checksum is just delivered to the server mover as part of the protocol (e.g. close operation for `dCap`), the transfer checksum has to be calcalated by the server mover on the fly. In order to communicate the different checksums to the embedding pool, the server mover has to implement the *ChecksumMover* interface in addition to the *MoverProtocol* Interface. A mover, not implementing the *MoverProtocol* is assumed not to handle checksums at all. The *Disk File Checksum* is calculated independedly of the mover within the pool itself.

```
public interface ChecksumMover {

        public void     setDigest( Checksum transferChecksum ) ;
        public Checksum getClientChecksum() ;
        public Checksum getTransferChecksum() ;

}
```

The pool will or will not call the *setDigest* method to advise the mover which checksum algorithm to use. If *setDigest* is not called, the mover is not assumed to calculate the *Transfer Checksum*.

```
java.security.MessageDigest transferDigest = transferChecksum.getMessageDigest() ;

                ***

        while( ... ){

                rc = read( buffer , 0 , buffer.length ) ;

                ***

                transferDigest.update( buffer , 0 , rc ) ;
        }
```

getClientChecksum and *getTransferChecksum* are called by the pool after the MoverProtocols runIO method has been successfully processed. These routines should return null if the corresponding checksum could not be determined for whatever reason.

```
public void  setDigest( Checksum transferChecksum ){

        this.transferChecksum = transferChecksum ;

        }
        public Checksum getClientChecksum(){
                return clientChecksumString == null ?
                        null :
                        Checksum( clientChecksumString ) ;
        }
        public Checksum getTransferChecksum(){ return transferChecksum ; }
```

The DCapProtocol_3_nio Mover

The *DCapProtocol_3_nio* mover implements the ChecksumMover interface and is able to report the *Client Checksum* and the *Transfer Checksum* to the pool. To enable the *DCapProtocol_3_nio* Mover to calculate the *Transfer Checksum*, either the cell context *dCap3-calculate-transfer-crc* or the cell batch line option *calcu-*

late-transfer-crc must be set to true. The latter may as well be set in the *.poolist file. *DCapProtocol_3_nio* disables checksum calculation as soon as the mover receives a client command except 'write' (e.g. read, seek or seek_and_write).

The ChecksumModule

The checksum module (as part of the Pool) and its command subset (*csm* ...) determines the behavious of the checksum calculation.

- `csm set policy -ontransfer=on`

  Movers, implementing the ChecksumMover interface, are requested to calculate the *Transfer Checksum*. Whether or not the mover actually performance the calculation might depend on additional, mover specific flags, like the *dCap3-calculate-transfer-crc* flag for the *DCapProtocol_3_nio* mover.

  If the mover reports the *Transfer Checksum* and there is a *Client Checksum* available, either from `pnfs` or from the mover protocol, the *Transfer Checksum* and the *Client Checksum* are compared. A mismatch will result in a *CRC Exception* .

  If there is no *Client Checksum* available whatsoever, the *Transfer Checksum* is stored in `pnfs`.

- `csm set policy -onwrite=on`

  After the dataset has been completely and successfully written to disk, the pool calculates the checksum based on the disk file (*Server File Checksum*). The result is compared to either the *Client Checksum* or the *Transfer Checksum* and a *CRC Exception* is thrown in case of a mismatch.

  If there is neither the *Client Checksum* nor the *Transfer Checksum* available, the *Server File Checksum* is stored in `pnfs`.

- `csm set policy -enforcecrc=on`

  In case of *-onwrite=off*, this options enforces the calculation of the *Server File Checksum* ONLY if neither the *Client Checksum* nor the *Transfer Checksum* has been sucessfully calculated. The result is stored in `pnfs`.

Migration Module

The migration module is a component of dCache pools introduced in version 1.9.1. The purpose of the component is essentially to copy or move the content of a pool to one or more other pools. The migration module replaces the copy manager found in previous releases of dCache. We advice against using the old copy manager, as it known to have problems.

Typical use cases for the migration module include:

- Vacating pools, that is, moving all files to other pools before decomisioning the pool.
- Caching data on other pools, thus distributing the load and increasing availability.
- As an alternative to the hopping manager.

Overview and Terminology

The migration module runs inside pools and hosts a number of migration jobs. Each job operates on a set of files on the pool on which it is executed and can copy or move those files to other pools. The migration module provides filters for defining the set of the files on which a job operates.

The act of copying or moving a file is called a migration task. A task selects a target pool and asks it to perform a pool to pool transfer from the source pool. The actual transfer is performed by the same component performing other pool to pool transfers. The migration module does not perform the transfer; it only orchestrates it.

The state of the target copy (the target state) as well as the source copy (the source state) can be explicitly defined. For instance, for vacating a pool the target state is set to be the same as the original source state, and the source state is changed to removed; for caching files, the target state is set to cached, and the source state is unmodified.

Sticky flags owned by the pin manager are never touched by a migration job, however the migration module can ask the pin manager to move the pin to the target pool. Care has been taken that unless the pin is moved by the pin manager, the source file is not deleted by a migration job, even if asked to do so. To illustrate this, assume a source file marked precious and with two sticky flags, one owned by foobar and the other by the pin manager. If a migration job is configured to delete the source file, but not to move the pin, the result will be that the file is marked cached, and the sticky flag owned by foobar is removed. The pin remains. Once it expires, the file is eligible for garbage collection.

All operations are idempotent. This means that a migration job can be safely rerun, and as long as everything else is unchanged, files will not be transferred a again. Because jobs are idempotent they do not need to maintain persistent state, which in turns means the migration module becomes simpler and more robust. Should a pool crash during a migration job, the job can be rerun and the remaining files will be transfered.

It is safe to run migration jobs while pools are in use. Once started, migration jobs run to completion and do only operate on those files that matched the selection filters at the time the migration job started. New files that arrive on the pool are not touched. Neither are files that change state after a migration job has been initialized, even though the selection filters would match the new state of the file. The exception to the rule is when files are deleted from the pool or change state such that they do no longer match the selection filter. Such files will be excluded from the migration job, unless the file was already processed. Rerunning a migration job will force it to pick up any new files. Because the job is idempotent, any files copied before are not copied again.

Permanent migration jobs behave differently. Rather than running to completion, permanent jobs keep running until explicitly cancelled. They monitor the pool for any new files or state changes, and dynamically add or remove files from the transfer queue. Permanent jobs are made persistent when the **save** command is executed and will be recreated on pool restart. The main use case for permanent jobs is as an alternative to using a central hopping manager.

Idempotence is achieved by locating existing copies of a file on any of the target pools. If an existing copy is found, rather than creating a new copy, the state of the existing copy is updated to reflect the target state specified for the migration job. Care is taken to never make a file more volatile than it already is: Sticky flags are added, or existing sticky flags are extended, but never removed or shortened; cached files may be marked precious, but not vice versa. One caveat is when the target pool containing the existing copy is

offline. In that case the existence of the copy cannot be verified. Rather than creating a new copy, the task fails and the file is put back into the transfer queue. This behaviour can be modified by marking a migration job as eager. Eager jobs create new copies if an existing copy cannot be immediately verified. As a rule of thumb, permanent jobs should never be marked eager. This is to avoid that a large number of unnecessary copies are created when several pools are restarted simultaneously.

A migration task aborts whenever it runs into a problem. The file will be reinserted at the end of the transfer queue. Consequently, once a migration job terminates, all files have been successfully transferred. If for some reason tasks for particular files keep failing, then the migration job will never terminate by itself as it retries indefinitely.

Command Summary

All commands begin with the string **migration**, e.g. **migration copy**. The commands **migration copy**, **migration cache** and **migration move** create new migration jobs. These commands take the same options and only differ in default values. Except for the number of concurrent tasks, transfer parameters of existing jobs cannot be changed. This is by design to ensure idempotency of jobs. The concurrency can be altered through the **migration concurrency** command.

Jobs are assinged a job ID and are executed in the background. The status of a job may be queried through the **migration info** command. A list of all jobs can be obtained through **migration ls**. Jobs stay in the list even after they have terminated. Terminated jobs can be cleared from the list through the **migration clear** command.

Jobs can be suspended, resumed and cancelled through the **migration suspend**, **migration resume** and **migration cancel** commands. Existing tasks are allowed to finish before a job is suspended or cancelled.

Examples

Vacating a pool

To vacate *sourcePool*, we first mark the pool read-only to avoid that more files are added to the pool, and then move all files to *targetPool*. It is not strictly necessary to mark the pool read-only, however if not done there is no guarantee that the pool is empty when the migration job terminates. The job can be rerun to move remaining files.

```
(sourcePool) admin > pool disable -rdonly
(sourcePool) admin > migration move targetPool
[1] RUNNING      migration move targetPool
(sourcePool) admin > migration info 1
Command    : migration move targetPool
State      : RUNNING
Queued     : 0
Attempts   : 1
Targets    : targetPool
Completed  : 0 files; 0 bytes; 0%
Total      : 830424 bytes
Concurrency: 1
Running tasks:
[0] 000100000000000000BFAE0: TASK.Copying -> [targetPool@local]
(sourcePool) admin > migration info 1
Command    : migration move targetPool
State      : FINISHED
Queued     : 0
```

```
Attempts   : 1
Targets    : targetPool
Completed  : 1 files; 830424 bytes
Total      : 830424 bytes
Concurrency: 1
Running tasks:
(sourcePool) admin > rep ls
(sourcePool) admin >
```

Caching recently accessed files

Say we want to cache all files belonging to the storage group atlas:default and accessed within the last month on a set of low-cost cache pools defined by pool group cache_pools. We can achieve this through the following command.

```
(sourcePool) admin > migration cache -target=pgroup -accessed=0..2592000 -storage=atlas:default
 cache_pools
[1] INITIALIZING migration cache -target=pgroup -accessed=0..2592000 -storage=atlas:default
 cache_pools
(sourcePool) admin > migration info 1
Command    : migration cache -target=pgroup -accessed=0..2592000 -storage=atlas:default cache_pools
State      : RUNNING
Queued     : 2577
Attempts   : 2
Targets    : pool group cache_pools, 5 pools
Completed  : 1 files; 830424 bytes; 0%
Total      : 2143621320 bytes
Concurrency: 1
Running tasks:
[72] 000100000000000000000BE10: TASK.Copying -> [pool_2@local]
```

The files on the source pool will not be altered. Any file copied to one of the target pools will be marked cached.

Renaming a Pool

A pool may be renamed with the following procedure, regardless of the type of files stored on it.

Disable file transfers from and to the pool with

```
(poolname) admin > pool disable -strict
```

Then make sure, no transfers are being processed anymore. All the following commands should give no output:

```
(poolname) admin > queue ls queue
(poolname) admin > mover ls
(poolname) admin > p2p ls
(poolname) admin > pp ls
(poolname) admin > st jobs ls
(poolname) admin > rh jobs ls
```

Now the files on the pools have to be unregistered on the `pnfs` server with

```
(poolname) admin > pnfs unregister
```

Even if the pool contains precious files, this is no problem, since we will register them again in a moment. The files might not be available for a short moment, though. Log out of the pool, and stop the service:

```
[root] # jobs/pool[1] -pool=poolDomainName stop
```

Rename the pool in the *poolDomain*.poollist-file. Restart the service:

```
[root] # jobs/pool -pool=poolDomainName -logfile=dCacheLocation/log/poolDomainNameDomain.log start
```

Register the files on the pool with

```
(poolname) admin > pnfs register
```

Pinning Files to a Pool

You may pin a file locally within the private pool repository:

```
(poolname) admin > rep set sticky pnfsid on|off
```

the 'sticky' mode will stay with the file as long as the file is in the pool. If the file is removed from the pool and recreated afterwards this information gets lost.

You may use the same mechanism globally: in the command line interface (local mode) there is the command

```
(poolname) admin > set sticky pnfsid
```

This command does:

1. Flags the file as sticky in the name space database (pnfs). So from now the filename is globally set sticky.
2. Will go to all pools where it finds the file and will flag it sticky in the pools.
3. All new copies of the file will become sticky.

[1] Filenames will always be relative to the dCache installation directory, which defaults to /opt/d-cache/.

Chapter 21. Migration of classic SE (nfs, disk) to dCache

This chapter contains a guide to migrate a classic SE to dCache.

The convertion of a classic SE to dCache is not complicated, but has to be done very carefully to prevent data losses.

We assume, that dCache is installed and configured.(the section called “Installing a Single Node dCache Instance”). To be on the safe side, we recommend to install a new pool on a different host, since there is no easy way to switch back to classic SE.

- create a new pool.
- for each file in the classic SE an entry in `pnfs` has to be created. then the file has to be moved to data directory in the pool control directory and the owner, group and size must be set in `pnfs`. To avoid mistakes we recomend to use a script developed and tested by the dCache developers. Run the script for each file which goes into dCache:

```
[root] #find . -type f -exec file2dcache.sh {} /pnfs/desy.de/data/fromSE /pool/pool1 \;
```

- start the pool. Since the pool has to recreate the inventory, the start up time will be longer than usually..
- connect to the dCache via admin interface and register the newly created files:

```
cd pool1
      pnfs register
      ..
      logoff
```

The newly migrated files shall be available already.

```
#!/bin/sh

if [ $# -ne 3 ]
then
   echo "Usage: $0 <file> <pnfs path> <pool base>"
   exit 1;
fi

SRC=$1
FILE=`basename $1`
DIR=`dirname $1`
PNFS_PRFIX=$2
POOL_BASE=$3

PNFS_FILE="${PNFS_PRFIX}/${DIR}/${FILE}"

if [ ! -f "${SRC}" ]
then
   echo "File ${SRC} do not exist."
   exit 1
fi
```

```
if [  -f "${PNFS_FILE}" ]
then
   echo "File ${PNFS_FILE}  already exist."
   exit 2
fi

if [ ! -d "${POOL_BASE}/control" ]
then
   echo "Creating directory [control]"
   mkdir ${POOL_BASE}/control
fi

if [ ! -d "${POOL_BASE}/data" ]
then
   echo "Creating directory [data]"
   mkdir ${POOL_BASE}/data
fi

if [ ! -f "setup" ]
then
   echo "Creating dummy [setup] file"
   touch  ${POOL_BASE}/setup
fi

echo "Creating file in pnfs"
if [ ! -d ${PNFS_PRFIX}/${DIR} ]
then
   mkdir -p ${PNFS_PRFIX}/${DIR} > /dev/null 2>&1
   if [ $? -ne 0 ]
   then
      echo "Failed to create directory ${PNFS_PRFIX}/${DIR}"
      exit 3;
   fi
fi

touch ${PNFS_FILE}
FILE_SIZE=`stat -c "%s" ${SRC}`
touch "${PNFS_PRFIX}/${DIR}/.(fset)(${FILE})(size)(${FILE_SIZE})"
chmod --reference=${SRC} ${PNFS_FILE}
chown --reference=${SRC} ${PNFS_FILE}

echo "Creating control file for pnfsID $PNFS_ID"
PNFS_ID=`cat "${PNFS_PRFIX}/${DIR}/.(id)(${FILE})"`
echo "precious" >  ${POOL_BASE}/control/${PNFS_ID}
echo "Copy ${SRC} to  ${POOL_BASE}/data/${PNFS_ID}"
cp  ${SRC}  ${POOL_BASE}/data/${PNFS_ID}

exit 0
```

Chapter 22. PostgreSQL and dCache

Vladimir Podstavkov

Mathias de Riese

Martin Radicke

PostgreSQL is used for various things in a dCache system: The *SRM*, the *pin manager*, the *space manager*, the *replica manager*, the *pnfs companion*, the *billing*, and the pnfs server might make use of one or more databases in a single or several separate PostgreSQL servers.

The SRM, the pin manager, the space manager, the replica manager, and the pnfs companion will use the PostgreSQL database as configured at cell start-up in the corresponding batch files. The billing will only write the accounting information into a database if it is configured with the option -useSQL. The pnfs server will use a PostgreSQL server if the pnfs-posgresql version is used. It will use several databases in the PostgreSQL server.

Installing a PostgreSQL Server

The preferred way to set up a PostgreSQL server should be the installation of the version provided by your OS distribution. It is strongly recommended to use version 8 or higher.

Install the PostgreSQL server, client and JDBC support with the tools of the operating system. You can download a suitable package from http://www.postgresql.org/ftp/. A version that is suitable for current versions of Scientific Linux 3 can be found at http://www.postgresql.org/ftp/binary/v8.1.0/linux/rpms/redhat/rhel-es-3.0/.

Initialize the database directory (usually /var/lib/pgsql/data/), start the database server, and make sure that it is started at system start-up. This may be done with

```
[root] # /etc/init.d/postgresql start
[root] # chkconfig postgresql on
```

If the start-up script does not perform the initialization automatically, it might have to be done with

```
[root] # initdb -D /var/lib/pgsql/data/
```

and the server is started manually with

```
[root] # postmaster -i -D /var/lib/pgsql/data/ >logfile 2>&1 &
```

The server has to be configured to admit TCP/IP connections from localhost. This is the default for version 8 of PostgreSQL.

For dCache version 1.6.6 release 1, please make sure the file /var/lib/pgsql/data/postgresql.conf contains

```
...
```

```
add_missing_from = on
...
```

This will not be necessary in future releases.

The file `/var/lib/pgsql/data/pg_hba.conf` should contain

```
...
local   all         all                           trust
host    all         all         127.0.0.1/32      trust
host    all         all         ::1/128           trust
```

Restart the server, e.g. with

```
[root] # /etc/init.d/postgresql restart
```

The configuration of the access rights in `/var/lib/pgsql/data/pg_hba.conf` is rather liberal: Any user on the local machine may connect to the PostgreSQL server as any database user without specifying a password. This way, you can be sure that problems will not be due to wrong access rights or passwords. See the section called “Configuring Access to PostgreSQL” of the dCache book for more advice on configuring PostgreSQL.

If a current version of PostgreSQL is not available for the distribution, it can be compiled as follows: You can download the source code from the official web site: http://www.postgresql.org/download and build it following the instruction: http://www.postgresql.org/docs/8.0/static/installation.html Here is a short version from that page:

```
[root] # ./configure --prefix=/usr/local/pgsql
[root] # gmake
[root] # su
[root] # gmake install
[root] # adduser postgres
[root] # mkdir /usr/local/pgsql/data
[root] # chown postgres /usr/local/pgsql/data
[root] # su - postgres
[root] # /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
[root] # /usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data > logfile 2>&1 &
[root] # /usr/local/pgsql/bin/createdb test
[root] # /usr/local/pgsql/bin/psql test
```

If there is another PostgreSQL installed on your machine, make sure `root`'s path is set so that executables, esp. **psql**, **postmaster**, and **pg_ctl**, are called from the V8.x.x PostgreSQL that you intend to use for dCache. (You will be doing most of this as `root`). Another thing is to make sure that the various `libpg.so` libraries are not invoked from the other PostgreSQL distribution. **`locate libpg.so`** will show which ones are on your system. Set `LD_LIBRARY` for `root` to insure that the `pnfs` PostgreSQL libs are seen first!

Configuring Access to PostgreSQL

In the installation guide instructions are given for configuring one PostgreSQL server on the admin node for all the above described purposes with generous access rights. This is done to make the installation as easy as possible. The access rights are configured in the file *`database_directory_name`*`/data/pg_hba.conf`. According to the installation guide the end of the file should look like

```
...
# TYPE  DATABASE    USER        IP-ADDRESS        IP-MASK           METHOD
local   all         all                                             trust
host    all         all         127.0.0.1/32                        trust
host    all         all         ::1/128                             trust
host    all         all         HostIP/32          trust
```

This gives access to all databases in the PostgreSQL server to all users on the admin host.

The databases can be secured by restricting access with this file. E.g.

```
...
# TYPE  DATABASE    USER        IP-ADDRESS        METHOD
local   all         postgres                      ident sameuser
local   all         pnfsserver                    password
local   all         all                           md5
host    all         all         127.0.0.1/32      md5
host    all         all         ::1/128           md5
host    all         all         HostIP/32          md5
```

The server is made aware of this with

```
[root] # pg_ctl reload
```

It can still be configured with the user `postgres`:

```
[root] # su - postgres
```

And the password for e.g. the user `pnfsserver` can be set with

```
[user] $ psql template1 -c "ALTER USER pnfsserver WITH PASSWORD 'yourPassword'"
```

The `pnfs` server is made aware of this password by changing the variable `dbConnectString` in the file `/usr/etc/pnfsSetup`:

```
...
export dbConnectString="user=pnfsserver password=yourPassword"
```

User access should be prohibited to this file with

```
[root] # chmod go-rwx /usr/etc/pnfsSetup
```

Performance of the PostgreSQL Server

On small systems it should never be a problem to use one single PostgreSQL server for all the functions listed above. In the standard installation, the `ReplicaManager`, the `pnfs` companion are not activated by default. The `billing` will only write to a file by default.

Whenever the PostgreSQL server is going to be used for another functionality, the impact on performance should be checked carefully. To improve the performance, the functionality should be installed on a separate host. Generally, a PostgreSQL server for a specific funcionality should be on the same host as the dCache cell accessing that PostgreSQL server, and the PostgreSQL server containing the databases for the `pnfs` server

should run on the same host as the `pnfs` server and the `PnfsManager` cell of the dCache system accessing it. Accordingly, the `pnfs` companion and the `pnfs` server itself will use the same PostgreSQL server.

It is especially useful to use a separate PostgreSQL server for the `billing` cell.

Note

The following is *work-in-progress*.

Create PostgreSQL user with the name you will be using to run `pnfs` server. Make sure it has `CREATEDB` privilege.

```
[user] $ psql -U postgres template1 -c "CREATE USER johndoe with CREATEDB"
[user] $ dropuser pnfsserver
[user] $ createuser --no-adduser --createdb --pwprompt pnfsserver
```

Table 22.1. Protocol Overview

Component	Database Host	Database Name	Database User	Database Password
SRM	`srmDatabaseHost`or if not set: `srmDbHost` or if not set: `localhost`	dcache	srmdcache	**`srmdcache`**
pin manag	`pinManagerDatabaseHost` or if not set: `srmDbHost` or if not set: `localhost`	dcache	srmdcache	**`srmdcache`**
`SpaceManager`	`spaceManagerDatabaseHost` or if not set: `srmDbHost` or if not set: `localhost`	dcache	srmdcache	**`srmdcache`**
companion	`companionDatabaseHost` or if not set: `localhost`	companion	srmdcache	**`srmdcache`**
`ReplicaManager`	`replicaManagerDatabaseHost` or if not set: `localhost`	replicas	srmdcache	**`srmdcache`**
`pnfs` server	`localhost`	admin, data1, exp0, ...	pnfsserver	--free--
billing	`billingDatabaseHost` or if not set: `localhost`	billing	srmdcache	**`srmdcache`**

Chapter 23. Complex Network Configuration

This chapter contains solutions for several non-trivial network configurations. The first section discusses the interoperation of dCache with firewalls and does not require any background knowledge about dCache other than what is given in the installation guide (Chapter 2, *Installing dCache*) and the first steps tutorial (Chapter 3, *Getting in Touch with dCache*). The following sections will deal with more complex network topologies, e.g. private subnets. Even though not every case is covered, these cases might help solve other problems, as well. Intermediate knowledge about dCache is required. Since most tasks require changes in the start-up configuration, the background information on how to configure the cell start-up, given in Chapter 5, *The Cell Package* will be useful.

Firewall Configuration

The components of a dCache instance may be distributed over several hosts (nodes). Some of these components are accessed from outside and consequently the firewall needs to be aware of that. This section assumes that all nodes are behind a firewall and have full access to each other. More complex setups are described in the following sections. Depending on the access method, certain ports have to be opened to only some of the nodes. We will describe the behaviour of a standard installation using the default values. Since dCache is very flexible, most port numbers may be changed in the configuration. The location (node) of any dCache component might also be changed from this standard.

The dCap Protocol

The `dCap` protocol should be used for local, trusted access only, because it is not authenticated. The traditional method is to mount `pnfs` locally on the client and use paths to the files on this local mount to address files. For this, the client has to be able to mount the NFS export of the `pnfs` server. It is also possible to use the `dCap` protocol with a URL instead of a local path within a `pnfs` mount. The URL has the form

```
dcap://dCapDoorHostFQN:dCapDoorPort/fullPnfsPath
```

If the *`dCapDoorPort`* is not specified, 22125 is used to establish a TCP connection to *`dCapDoorHostFQN`* (see next paragraph). In this case no NFS mount is needed anymore. However, the access is unauthenticated and therefore access is only granted if the “other” part of the UNIX rights are set accordingly. In other words: The user is mapped to nobody for unauthenticated `dCap` access.

In both cases (`pnfs` mount and URL access) the `dCap` client (`dCap` library or **dccp** command) will connect to the `dCap` door (`doorDomain`) on the admin node with TCP port 22125 (can be changed in `config/dCacheSetup` with `dCapPort`). After the pool manager selected a pool to be used for the transfer (the section called “The Pool Selection Mechanism” describes how to configure that selection mechanism.) this pool will establish the data connection to the client on a TCP port which has been selected by the client. The port range to use may be specified on the client side (e.g. by the `-p` option of the **dccp** command.)

The GSIdCap Protocol

The GSIdCap protocol is the dCap protocol with a GSI authentication wrapper (tunnel). The mechanism is the same as described for the URL-stype dCap protocol in the previous section. The only difference is the default port number: For the GSIdCap door the default port number is 22128. It is specified in config/dCacheSetup with the parameter dCapGsiPort.

Another difference between the dCap door and GSIdCap doors is that the dCap door is started on the admin node and there can only be one in the standard configuration, while there may be several GSIdCap doors on separate nodes. Correspondingly, ports have to be opened in a firewall. Note that it should not be necessary to run as many GSIdCap doors as GridFTP doors (see below), because no data is transfered through the GSIdCap door.

The GridFTP Protocol

A GridFTP client connects to one of the GridFTP doors on TCP port 2811. The data connections are established independent of the direction of the data transfer. In “active” FTP mode the server connects to the client while in “passive” FTP mode the client connects to the server.

In “active” FTP mode the pool selected by the pool manager (see the section called “The Pool Selection Mechanism”) will open one or more data connections to the client on a TCP port in the range between 20000 and 25000. In “passive” FTP mode, the client will establish the data connections to the GridFTP door in the same port range. The pool will connect to the door and the door will route the data traffic. It is not possible to establish a direct connection between pool and client in “passive” mode, because the FTP protocol redirection mechanism has to be triggered before the client sends the name of the requested file.

The SRM Protocol

An SRM is a webservice which uses the https as transport protocol and negotiates data transfers between the client and the server as well as between the server and another server. For the actual data transfer one of the other protocols is negotiated. Usually this is GridFTP - especially for wide-area transfers. There are two things to note about SRM-initiated GridFTP transfers:

For reading data, only “active” FTP mode will be used, i.e. the pool containing the data will connect to the client or to the server which should receive the data. For writing, only “passive” FTP mode will be used, i.e. the client or the pool containing the data will connect to the client or to the server which should receive the data.

Apart from SRM put and get operations which always copy data between one SRM and the client there is also a true SRM copy from one SRM to another SRM. There are two modes for SRM copy: “pull” and “push” mode. If the destination SRM is dCache based and SRM pull mode (default) is used, the destination pool will play the role of the GridFTP client, will contact the GridFTP door of the source instance and receive the data directly from the source pool (if the source system is a dCache system). If push mode is used and the source is a dCache based SRM, the source pool will be the GridFTP client and will send the data to the GridFTP door of the destination. All this might have to be considered when designing the system and configuring the firewall.

Pool Selection

Restricting wide-area transfers to a subset of your dCache pools may be done with the *pool selection unit* in the pool manager. the section called “The Pool Selection Mechanism” contains a describtion on how to do that. This can be useful to ease firewall configurations, optimize throughput, and improve security.

Protocol Overview

The following table gives an overview about the default ports used by the client protocols supported by dCache. Note that all of them may be changed in `config/dCacheSetup`.

Table 23.1. Protocol Overview

Protocol	Port(s)	Direction	Nodes
`dCap`	22125	incoming	doorDomain (admin node)
	any	outgoing	pools
`GSIdCap`	22128	incoming	gsidcapDomain (where GSIDCAP=yes in node_config)
	any	outgoing	pools
`GridFTP`	2811	incoming	gridftpDomain (where GRIDFTP=yes in node_config)
	20000-25000	outgoing (active `FTP`)	pools
	20000-25000	incoming (passive `FTP`)	gridftpDomain
`SRM`	8443	incoming	srmDomain

`GridFTP` Connections via two or more Network Interfaces

Description

The host on which the `GridFTP` door is running has several network interfaces and is supposed to accept client connections via all those interfaces. The interfaces might even belong to separate networks with no routing from one network to the other.

As long as the data connection is opened by the `GridFTP` server (active FTP mode), there is no problem with having more than one interface. However, when the client opens the data connection (passive FTP mode), the door (FTP server) has to supply it with the correct interface it should connect to. If this is the wrong interface, the client might not be able to connect to it, because there is no route or the connection might be inefficient.

Also, since a `GridFTP` server has to authenticate with an SSL grid certificate and key, there needs to be a separate certificate and key pair for each name of the host. Since each network interface might have a

different name, several certificates and keys are needed and the correct one has to be used, when authenticating via each of the interfaces.

Solution

Start a separate `GridFTP` server cell on the host for each interface on which connections should be accepted.

The cells may be started in one domain or in separate domains. The cells have to have different names, since they are *well known* cells. Each cell has to be configured, only to listen on the interface it should serve with the `-listen` option. The locations of the grid host certificate and key files for the interface have to be specified explicitly with the `-service-cert` and `-service-key` options.

The following example shows a setup for two network interfaces with the hostnames `door-a.grid.domain` (111.111.111.5) and `door-b.other.domain` (222.222.222.5) which are served by two `GridFTP` door cells in one domain:

Example 23.1. Batch file for two `GridFTP` doors serving separate network interfaces

```
set printout default 2
set printout CellGlue none
onerror shutdown
check -strong setupFile
copy file:${setupFile} context:setupContext
import context -c setupContext
check -strong serviceLocatorPort serviceLocatorHost
check -strong sshPort ftpPort
create dmg.cells.services.RoutingManager  RoutingMgr
create dmg.cells.services.LocationManager lm \
       "${serviceLocatorHost} ${serviceLocatorPort}"

create dmg.cells.services.login.LoginManager GFTP-door-a \
            "2811 \
      -listen=111.111.111.5 \
             -export \
             diskCacheV111.doors.GsiFtpDoorV1 \
             -prot=raw \
             -service-cert=/etc/grid-security/door-a.grid.domain-cert.pem \
             -service-key=/etc/grid-security/door-a.grid.domain-key.pem \
             ..
             ..
"

create dmg.cells.services.login.LoginManager GFTP-door-b \
            "2811 \
             -listen=222.222.222.5 \
             -export \
             diskCacheV111.doors.GsiFtpDoorV1 \
             -prot=raw \
             -service-cert=/etc/grid-security/door-b.other.domain-cert.pem \
             -service-key=/etc/grid-security/door-b.other.domain-key.pem \
             ..
             ..
"
```

This batch file is very similar to the batch file for the `GridFTP` door in the standard setup. (Comments have been left out.) It only contains an additional create command for the second cell and the emphasized changes within the two create commands: The cell names, the `-listen` option with the IP address of the corresponding interface and the `-service-cert` and `-service-key` options with the host certificate and key files.

GridFTP with Pools in a Private Subnet

Description

If pool nodes of a dCache instance are connected to a *secondary interface* of the GridFTP door, e.g. because they are in a private subnet, the GridFTP door will still tell the pool to connect to its primary interface, which might be unreachable.

The reason for this is that the control communication between the door and the pool is done via the network of TCP connections which have been established at start-up. In the standard setup this communication is routed via the dCache domain. However, for the data transfer, the pool connects to the GridFTP door. The IP address it connects to is sent by the GridFTP door to the pool via the control connection. Since the GridFTP door cannot find out which of its interfaces the pool should use, it normally sends the IP address of the *primary interface*.

Solution

Tell the GridFTP door explicitly which IP it should send to the pool for the data connection with the `-ftp-adapter-internal-interface` option. E.g. if the pools should connect to the secondary interface of the GridFTP door host which has the IP address `10.0.1.1`, the following batch file would be appropriate:

Example 23.2. Batch file for two GridFTP doors serving separate network interfaces

```
set printout default 2
set printout CellGlue none
onerror shutdown
check -strong setupFile
copy file:${setupFile} context:setupContext
import context -c setupContext
check -strong serviceLocatorPort serviceLocatorHost
check -strong sshPort ftpPort
create dmg.cells.services.RoutingManager  RoutingMgr
create dmg.cells.services.LocationManager lm \
       "${serviceLocatorHost} ${serviceLocatorPort}"

create dmg.cells.services.login.LoginManager GFTP \
            "2811 \
             -export \
             diskCacheV111.doors.GsiFtpDoorV1 \
             -prot=raw \
             -clientDataPortRange=${clientDataPortRange} \
            -root=${ftpBase} \
             -kpwd-file=${kpwdFile} \
             -tlog=/tmp/dcache-ftp-tlog \
             -maxLogin=100 \
             -brokerUpdateTime=5 \
             -protocolFamily=gsiftp \
             -loginBroker=LoginBroker \
             -poolManagerTimeout=5400 \
             -pnfsTimeout=120 \
             -maxRetries=80 \
             -maxStreamsPerClient=10 \
     -ftp-adapter-internal-interface=10.0.1.1 \
"
```

This batch file is very similar to the batch file for the GridFTP door in the standard setup. (Comments have been left out.) The emphasized last line has the desired effect.

Doors in the DMZ

Description

Some doors - e.g. for grid access - are located in the DMZ while the rest of the dCache instance is in the intranet. The firewall is configured in such a way that the doors cannot reach the location manager (usually on the admin node together with the pool manager) via port 11111 (or as configured in the variable `serviceLocatorPort` in `config/lmSetup`).

Solution

Please contact `<support@dcache.org>` if you need a solution for this problem.

Chapter 24. Accounting

The raw information about all dCache activities can be found in `billing/YYYY/MM/billing-YYYY.MM.DD`.[1] A typical line looks like

```
05.31 22:35:16 [pool:pool-name:transfer] [000100000000000000001320,24675] myStore:STRING@osm 24675
 474 true {GFtp-1.0 client-host-fqn 37592} {0:""}
```

The first bracket contains the pool name, the second the *pnfs ID* and the size of the file which is transferred. Then follows the *storage class*, the actual amount of bytes transferred, and the number of milliseconds the transfer took. The next entry is `true` if the transfer was a wrote data to the pool. The first braces contain the protocol, client FQN, and the client host data transfer listen port. The final bracket contains the return status and a possible error message.

The dCache web interface (described in the section called “The Web Interface for Monitoring dCache”) contains under the menue point “Actions Log” summary information extracted from the information in the `billing`-directory.

The accounting information can also be redirected into a database. When interested in this feature, please contact <support@dcache.org>.

[1] Filenames will always be relative to the dCache installation directory, which defaults to `/opt/d-cache/`.

Chapter 25. Protocols

dCap options mover and client options

Patrick Fuhrmann
Tigran Mkrtchyan

`dCap` is the native random access I/O protocol for files within dCache. In addition to the usual data transfer mechanisms, it supports all necessary file metadata and name space manipulation operations.

In order to optimize I/O transferrates and memory consumption `dCap` allows to configure parameters within the client and the server. e.g:

- TCP Socket send and receive buffer sizes.
- I/O buffers sizes.

TCP send/recv buffer sizes from the servers point of view

There are two parameters per I/O direction, determining the actual TCP send/recv buffer size used for each transfer. Those values can be set within the `config/pool.batch` file on the pool nodes.

- `defaultSend/RecvBufferSize` : this value is used if the dCap client doesn't try to set this value. The default value for this parameter is 256K Bytes.
- `maxSend/RecvBufferSize` : this value is the maximum value, the mover is allowed to use. It's used if either the `defaultSend/RecvBufferSize` is larger or the client asks for a larger value. The default value for this parameter is 1MBytes.

On the server side, the max/defaultSend/RecvBuffer value can either be set in the `config/pool.batch` file or in the `config/*.poollist` files.

Using the batch context :

```
set context dCap3-maxSendBufferSize value in bytes
set context dCap3-maxRecvBufferSize value in bytes
set context dCap3-defaultSendBufferSize value in bytes
set context dCap3-defaultRecvBufferSize value in bytes
```

Or it may specified in the create ... command line

```
  create diskCacheV111.pools.MultiProtocolPool2 ${0} \
  "!MoverMap \
  ${1} \
  -defaultSendBufferSize=value in bytes \
  *** \
  -${2} -${3} -${4} -${5} -${6} -${7} -${8} \
"
```

The most appropriate way to specify those values on the server side is certainly to add the corresponding entry in the `config/...poollist`. The entry would look like

```
dcache30_1  /dcache/pool  sticky=allowed maxSendBufferSize=value in bytes tag.hostname=dcache30 ***
```

Please note the different ways of using the '=' and the '-' sign in the different alternatives.

TCP send/recv buffer sizes from the dCap clients point of view

For a full list of dCap library API calls and dccp options, please refer to to `http://www.dcache.org/manuals/libdcap.shtml` and `http://www.dcache.org/manuals/dccp.shtml` respectively. To set the local and remote TCP buffer send and receive buffers either use the API call `dc_setTCPSend/ReceiveBuffer(int size)` or the `-r SIZE -s SIZE` dccp options. In both cases the value is transferred to the remote mover which tries to set the corresponding values. Please not the the server protects itself by having a maximum size for those values which it doesn't exceed. Please check the section 'TCP send/recv buffer sizes from the servers point of view' to learn how to change those values.

Specifying dCap open timeouts

Patrick Fuhrmann

In cases where dccp/dcap requests a file which is still on tertiary storage, the user resp. the administrator might what to limit the time, dccp/dCap waits in the open call until the file has been fetched from backend storage. This, so called `openTimeout`, can be specified on the server or on the client. In all cases the `-keepAlive` must be specified with an appropriate number of seconds on the cell create command in the door batch files. The following mechanisms are available to specify open timeouts :

Table 25.1. Open Timeout mechanisms

Precedence	Mechanism	Key Name	Example
Lowest	context	dCap-openTimeout	set context dCap-openTimeout 200
...	context	openTimeout	set context openTimeout 200
...	cell create command line	openTimeout	-openTimeout=200
Highest	dccp command line	-o	dccp -o200 SOURCE DESTINATION

```
#
#    dCap    D o o r (create command line example)
#
create dmg.cells.services.login.LoginManager DCap-2 \
            "${specialDCapPort} \
             diskCacheV111.doors.DCapDoor \
             -export \
             *** \
             -keepAlive=60 \
             -openTimeout=300 \
             *** \
```

```
             -loginBroker=LoginBroker"
```

```
#
#    dCap    D o o r (context example)
#
set context dCap-openTimeout 200
#
create dmg.cells.services.login.LoginManager DCap-2 \
            "${specialDCapPort} \
             diskCacheV111.doors.DCapDoor \
             -export \
             *** \
             -keepAlive=60 \
             *** \
             -loginBroker=LoginBroker"
```

```
[user] $ dccp -o200 /pnfs/desy.de/data/dteam/private/myfile /dev/null
```

If the openTimeout expires while a read transfer is already active, this transfer will be interrupted, but it will automatically resume because the client can't destinguish between a network failure and a timeout. So the timeout disturbes the read but it will finally succeed. This is different for write. If a write is interrupted by a timeout in the middle of a transfer, dccp will stuck. (This is not a feature and needs further investigation).

Using the dCap protocol for strict file checking

Patrick Fuhrmann
Tigran Mkrtchyan

The `dCap` protocol allows to check whether a dataset is on tape only or has a copy on a dCache disk. The `dCap` library API call is `int dc_check(const char *path, const char *location)` and the dccp options are `-t -1 -P`. For a full list of dCap library API calls and dccp options, please refer to to `http://www.dcache.org/manuals/libdcap.shtml` and `http://www.dcache.org/manuals/dccp.shtml` respectively. Using a standard dCache installation those calls will return a guess on the file location only. It is neither checked whether the file is really on that pool or if the pool is up. To get a strict checking a `dCap door` has to be started with a special (-check=strict) option.

```
#
#    dCap    D o o r
#
create dmg.cells.services.login.LoginManager DCap-strict \
            "${specialDCapPort} \
             diskCacheV111.doors.DCapDoor \
             -check=strict \
             -export \
             -prot=telnet -localOk \
             -maxLogin=1500 \
             -brokerUpdateTime=120 \
             -protocolFamily=dcap \
             -loginBroker=LoginBroker"
```

This door will do a precise checking (-check=strict). To get the dCap lib and dccp to use this door only, the `DCACHE_DOOR` environment variable has to be set to `doorHost:specialDCapPort` in the shell,

dccp is going to be used. In the following example we assume that the `specialDCapPort` has been set to `23126` :

```
[user] $ export DCACHE_DOOR=dcachedoorhost:23126
[user] $ dccp -P -t -1 /pnfs/domain.tv/data/cms/users/waste.txt
```

If **dccp** returns `File is not cached` and this dCache instance is connected to an HSM, the file is no longer on one of the dCache pools but is assumed to have a copy within the HSM. If the **dccp** returns this message and no HSM is attached, the file is either on a pool which is currently down or the file is lost.

Passive dCap

Tigran Mkrtchyan
Patrick Fuhrmann

The `dCap` protocol, similiar to `FTP`, uses a control channel to request a transfer which is subsequently done through data channels. Per default, the data channel is initiated by the server, connecting to an open port in the client library. This is commonly known as active transfer. Starting with dCache 1.7.0 the `dCap` protocol supports passive transfer mode as well, which consequently means that the client connects to the server pool to initiate the data channel. This is essential to support `dCap` clients running behind firewalls and within private networks.

Preparing the server for dCap passive transfer

The port(s), the server pools should listens on, can be specified by the `org.dcache.net.tcp.portrange` variable, as part of the 'java_options' directive in the `config/dCacheSetup` configuration file. A range has to be given if pools are split amoung multiple JVMs. E.g:

```
java_options="-server ... -Dorg.dcache.dcap.port=0 -Dorg.dcache.net.tcp.portrange=33115:33145"
```

Switching the dCap library resp. dccp to PASSIVE

Note

The commonly used expression 'passive' is seen from the server perspective and actually means 'server passive'. From the client perspective this is of course 'active'. Both means that the client connects to the server to establish the data connection. This mode is supported by the server starting with 1.7.0 and dccp with 1-2-40 (included in 1.7.0)

The following `dCap` API call switches all subsequent dc_open calls to server-passive mode if this mode is supported by the corresponding door. (dCache Version >= 1.7.0).

```
void dc_setClientActive()
```

The environment variable `DCACHE_CLIENT_ACTIVE` switches the `dCap` library to server-passive. This is true for `dCap`, `dCap` preload and **dccp**.

dccp switches to server-passive when issuing the **-A** command line option.

Access to SRM and GridFTP server from behind a firewall

Timur Perelmutov
Mathias de Riese

This describes firewall issues from the clients perspective. the section called “Firewall Configuration” discusses the server side.

When files are transferred in GridFTP active mode from GridFTP server to the GridFTP client, server establishes data channel(s) by connecting to the client. In this case client creates a TCP socket, bound to some particular address on the client host, and sends the client host IP and port to the server. If the client host is running a firewall, firewall might refuse server’s connection to the client’s listening socket. Common solution to this problem is establishing a range of ports on the client’s host that are allowed to be connected from Internet by changing firewall rules.Once the port range is defined the client can be directed to use one of the ports from the port ranges when creating listening tcp sockets.

Access with srmcp

If you are using **srmcp** as a client you need to do the following:

- create a directory $HOME/.globus if it does not exist.
- create and/or edit a file $HOME/.globus/cog.properties by appending a new line reading

```
tcp.port.range=min,max
```

  where *min* and *max* are the lower and upper bounds of the port range.

With the latest **srmcp** release you can use the globus_tcp_port_range option:

```
[user] $ srmcp -globus_tcp_port_range=minValue:maxValue ...
```

A range of ports open for TCP connections is specified as a pair of positive integers separated by ":". This is not set by default.

Access with globus-url-copy

If you are transferring files from gridftp server using **globus-url-copy**, you need to define an environment variable GLOBUS_TCP_PORT_RANGE, in the same shell in which **globus-url-copy** will be executed.

In sh/bash you do that by invoking the following command:

```
[user] $ export GLOBUS_TCP_PORT_RANGE="min,max"
```

in csh/tcsh you invoke:

```
[user] $ setenv GLOBUS_TCP_PORT_RANGE "min,max"
```

here *min* and *max* are again the lower and upper bounds of the port range

Disableing unauthenticated dCap via SRM

In some cases SRM transfers fail because they are tried via the plain dCap protocol (URL starts with dcap://). Since plain dCap is unauthenticated, the dCache server will have no information about the user trying to access the system. While the transfer will succeed if the UNIX file permissions allow access to anybody (e.g. mode 777), it will fail otherwise.

Usually all doors are registered in SRM as potential access points for dCache. During a protocol negotiation the SRM chooses one of the available doors. You can force **srmcp** to use the GSIdCap protocol (-protocol=gsidcap) or you can unregister plain, unauthenticated dCap from known protocols: From the file config/door.batch remove -loginBroker=LoginBroker and restart dCap door with

```
[root] # jobs/door stop
[root] # jobs/door -logfile=dCacheLocation/log/door.log start
```

Chapter 26. Advanced Tuning

The use cases described in this chapter are only relevant for large-scale dCache instances which require special tuning according to a longer experience with client behaviour.

Multiple Queues for Movers in each Pool

Description

Client requests to a dCache system may have rather diverse bahaviour. Sometimes it is possible to classify them into several typical usage patterns. An example are the following two usage patterns:

Example 26.1. To Concurrent Usage Patterns

Data is copied with a high transfer rate to the dCache system from an external source. This is done via the `GridFTP` protocol. At the same time batch jobs on a local farm process data. Since they only need a small part of each file, they use the `dCap` protocol via the `dCap` library and seek to the position in the file they are interested in, read a few bytes, do a few hours of calculations, and finally read some more data.

As long as the number of active requests do not exceed the maximum number of allowed active requests, the two types of requests are processed concurrently. The `GridFTP` transfers complete at a high rate while the processing jobs take hours to finish. This maximum number of allowed requests is set with **mover set max active** and should be tuned according to capabilities of the pool host.

However, if requests are queued, the slow processing jobs might clog up the queue and not let the fast `GridFTP` request through, even though the pool just sits there waiting for the processing jobs to request more data. While this could be temporarily remedied by setting the maximum active requests to a higher value, then in turn `GridFTP` request would put a very high load on the pool host.

The above example is pretty realistic: As a rule of thumb, `GridFTP` requests are fastest, `dCap` requests with the **dccp** program are a little slower and `dCap` requests with the `dCap` library are very slow. However, the usage patterns might be different at other sites and also might change over time.

Solution

Use separate queues for the movers, depending on the door initiating them. This easily allows for a separation of requests of separate protocols. Up to 10 mover queues for client transfers are available since dCache version 1.6.6. Earlier versions support only one queue. (Transfers from an to a *tape backend* and *pool-to-pool transfers* are handled by separate queues, one for each of these transfers.)

A finer grained queue selection mechanism based on, e.g. the `IP` address of the client or the file which has been requested, is not possible with this mechanism. However, the *pool selection unit (PSU)* may provide a separation onto separate pools using those criteria.

In the above example, two separate queues for fast `GridFTP` transfers and slow `dCap` library access would solve the problem. The maximum number of active movers for the `GridFTP` queue should be set to a lower

value compared to the `dCap` queue since the fast `GridFTP` transfers will put a high load on the system while the `dCap` requests will be mostly idle.

Configuration

For a multi mover queue setup, the pools have to be told to start several queues and the doors have to be configured to use one of these. It makes sense to create the same queues on all pools. This is done by the following change to the `config/pool.batch` file:

Example 26.2. Modified `config/pool.batch` file for multiple mover queues

```
...
define context startPools endDefine
  create diskCacheV111.pools.MultiProtocolPool2 ${0} \
         "!MoverMap \
         ${1} \
         -io-queues=queueName-1[,queueName-2[,...,queueName-10]] \
         -recover-control=yes \
         -version=4 \
         -sticky=allowed \
         -sendHitInfoMessages=yes \
         -${2} -${3} -${4} -${5} -${6} -${7} -${8} \
         "
endDefine
...
```

The same can be achived by appending `-io-queues=`*`queueName-1,...,queueName-n`* to each line in the *`poollist`* *file*. However, this only makes sense if the pools should not all have the same queues.

The first in this list of queues (*`queueName-1`*) is the *default mover queue*. Transfers not requesting a particular mover queue or requesting a mover queue not existing on the selected pool, are handled by this default queue.

The pool cell commands **mover ls** and **mover set max active** have an `-queue` option to select the mover queue to operate on. Without this option, **mover set max active** will act on the default queue while **mover ls** will list the requests of all pools for backward compatibility.

Each door may be configured to use a particular mover queue. The pool, selected for this request, doesn't depend on the selected mover queue. So a request may go to a pool which doesn't have the particular mover queue configured and will consequently end up in the default mover queue of that pool.

The doors are configured to use a particular mover queue as in the following example:

Example 26.3. Batch file for a GridFTP door using a mover queue

```
...
create dmg.cells.services.login.LoginManager GFTP \
       "portName \
       diskCacheV111.doors.GsiFtpDoorV1 \
       -io-queue=queueName \
       ... \
"
```

All requests send from this door will ask to be scheduled to the given mover queue. The selection of the pool is not affected.

For the dCap protocol, the corresponding door may be configured to allow the client to determine the mover queue name. In that case the client may use the extra option facility to specify a mover queue. Whether the the dCap door allows the client to request a particular mover queue or not is configured with the `-io-queue={allowed|denied}` option as in the following example:

Example 26.4. Batch file for a dCap door for allowing the client to select the mover queue

```
...
create dmg.cells.services.login.LoginManager DCap \
       "${dCapPort} \
       diskCacheV111.doors.DCapDoor \
       -io-queue=queueName \
       -io-queue-overwrite=allowed \
       ... \
"
```

With the **dccp** command the queue can now be specified as follows:

```
[user] $ dccp -X-io-queue=queueName source destination
```

Since **dccp** requests may be quite different from other requests with the dCap protocol, this feature may be used to use separate queues for **dccp** requests and other dCap library requests. Therefore, the **dccp** command may be changed in future releases to request a special **dccp**-queue by default.

Tunable Parameters

gridftp

Table 26.1. Variable Overview

Variable	Default Value	Description
gsidcapIoQueue	*Not set*	GSIdCap I/O queue name
dcapIoQueue	*Not set*	Insecure dCap I/O queue name
gsidcapIoQueueOverwrite	denied	Is application allowed to overwrite queue name?
dcapIoQueueOverwrite	denied	Is application allowed to overwrite queue name?

GridFTP

Table 26.2. Variable Overview

Variable	Default Value	Description
gsiFtpPortNumber	2811	GSI-FTP port listen port
spaceReservation	*False*	Use the space reservation service
spaceReservationStrict	*False*	Use the space reservation service
performanceMarkerPeriod	180	Performance markers in seconds

Variable	Default Value	Description
gplazmaPolicy	${ourHomeDir}/etc/dcachesrm-gplazma.policy	Location of the gPlazma Policy File
useGPlazmaAuthorizationModule	*False*	Use the gPlazma module
useGPlazmaAuthorizationCell	*True*	Use the `gPlazma` cell
gsiftpPoolManagerTimeout	5400	Pool Manager timeout in seconds
gsiftpPoolTimeout	600	Pool timeout in seconds
gsiftpPnfsTimeout	300	Pnfs timeout in seconds
gsiftpMaxRetries	80	Number of PUT/GET retries
gsiftpMaxStreamsPerClient	10	Number of parallel streams per `FTP` PUT/GET
gsiftpDeleteOnConnectionClosed	*True*	Delete file on connection closed
gsiftpMaxLogin	100	Maximum number of concurrently logged in users
gsiftpAdapterInternalInterface	*Not set*	In case of two interfaces
clientDataPortRange	20000:25000	The client data port range
kpwdFile	`${ourHomeDir}/etc/dcache.kpwd`	Legacy authorization

SRM

Table 26.3. Variable Overview

Variable	Default Value	Description
srmPort	8443	srmPort
srmDatabaseHost	`localhost`	srmDatabaseHost
srmTimeout	3600	srmTimeout
srmVacuum	*True*	srmVacuum
srmVacuumPeriod	21600	srmVacuumPeriod
srmProxiesDirectory	`/tmp`	srmProxiesDirectory
srmBufferSize	1048576	srmBufferSize
srmTcpBufferSize	1048576	srmTcpBufferSize
srmDebug	*True*	srmDebug
srmGetReqThreadQueueSize	1000	srmGetReqThreadQueueSize
srmGetReqThreadPoolSize	100	srmGetReqThreadPoolSize
srmGetReqMaxWaitingRequests	1000	srmGetReqMaxWaitingRequests
srmGetReqReadyQueueSize	1000	srmGetReqReadyQueueSize
srmGetReqMaxReadyRequests	100	srmGetReqMaxReadyRequests
srmGetReqMaxNumberOfRetries	10	srmGetReqMaxNumberOfRetries
srmGetReqRetryTimeout	60000	srmGetReqRetryTimeout
srmGetReqMaxNumOfRunningBySameOwner	10	srmGetReqMaxNumOfRunningBySameOwner
srmPutReqThreadQueueSize	1000	srmPutReqThreadQueueSize
srmPutReqThreadPoolSize	100	srmPutReqThreadPoolSize
srmPutReqMaxWaitingRequests	1000	srmPutReqMaxWaitingRequests
srmPutReqReadyQueueSize	1000	srmPutReqReadyQueueSize
srmPutReqMaxReadyRequests	100	srmPutReqMaxReadyRequests
srmPutReqMaxNumberOfRetries	10	srmPutReqMaxNumberOfRetries
srmPutReqRetryTimeout	60000	srmPutReqRetryTimeout
srmPutReqMaxNumOfRunningBySameOwner	10	srmPutReqMaxNumOfRunningBySameOwner

Variable	Default Value	Description
srmCopyReqThreadQueueSize	1000	srmCopyReqThreadQueueSize
srmCopyReqThreadPoolSize	100	srmCopyReqThreadPoolSize
srmCopyReqMaxWaitingRequests	1000	srmCopyReqMaxWaitingRequests
srmCopyReqMaxNumberOfRetries	30	srmCopyReqMaxNumberOfRetries
srmCopyReqRetryTimeout	60000	srmCopyReqRetryTimeout
srmCopyReqMaxNumOfRunningBySameOwner	10	srmCopyReqMaxNumOfRunningBySameOwner

Chapter 27. Statistics Module for pre 1.6.7 releases

Patrick Fuhrmann

General remarks

Purpose

The dCache statistics module collects information on the amount of data stored on all pools and the total data flow including streams from and to tertiary storage systems. The module produces an ASCII file once per hour, containing a table with information on the amount of used disk space and the data transferred starting midnight up the this point in time. Data is sorted per pool and storage class. In addition to the hourly statistics, files are produced reporting on the daily, monthly and yearly dCache activities. If enabled, a 'html' tree is produced and updated once per hour allowing to navigate through the collected statistics information.

Availability

The dCache statistics module will be part of dCache releases 1.6.7 and higher. The code is part of 1.6.6 but needs to be enabled. At the end of this chapter some advise is given on how to do that.

Directory and File formats

Directory Structure

The statistics module automatically creates a directory tree, structured according to years, months and days. Once per hour, a `total.raw` file is produced underneath the active `year`, `month` and `day` directories, containing the sum over all pools and storage classes of the corresponding time interval. A `days` directory contains detailed statistics per hour and for the whole day.

```
/StatBase/YYYY/total.raw
/StatBase/YYYY/MM/total.raw
/StatBase/YYYY/MM/DD/total.raw
/StatBase/YYYY/MM/DD/YYYY-MM-DD-day.raw
/StatBase/YYYY/MM/DD/YYYY-MM-DD-HH.raw
```

File Format

Format of `YYYY-MM-DD-HH.raw` or `YYYY-MM-DD-day.raw` files.

Table 27.1. File Format

Column Number	Column Description
0	Pool Name

Column Number	Column Description
1	Storage Class
2	Bytes stored on this pool for this storage class at beginning of day
3	Number of files stored on this pool for this storage class at beginning of day
4	Bytes stored on this pool for this storage class at this hour or end of day
5	Number of files stored on this pool for this storage class at this hour or end of day
6	Total Number of transfers (in and out, dCache-client)
7	Total Number of restores (HSM to dCache)
8	Total Number of stores (dCache to HSM)
9	Total Number errors
10	Total Number of bytes transferred into dCache (from clients)
11	Total Number of bytes transferred out of dCache (to clients)
12	Total Number of tranferred from HSM to dCache
13	Total Number of tranferred from dCache to HSM

HTML Directory Structure

In case the HTML functionality is enabled, which is the default, the statistics modules creates an `html` tree allowing to navigate between the different statistics views. Those files populate the same directory structure as the `xxx.raw` files. The HTML root is at:

```
/StatBase/index.html
```

and may be accessed via the dCache `HTTP` services using

```
http://headnode:2288/statistics/
```

(Don’t forget the trailing slash)

How to activate the statistics module in 1.6.6

General remarks

The statistics module collects parts of its information in memory of cells within the `httpDomain` and the `statisticsDomain`. Consequently this information is lost if one or all of those components are restarted. As a result, the day statistics may be significantly wrong if the restart happens at the end of the day. We hope to overcome this problem with 1.6.7 and higher.

Moreover, because the module can only add up pool space for those pools which are up during the inquery phase, disk space of pools which are down during that time is not counted.

How to activate the statistics module in 1.6.6

Create a file in the dCache config directory with the following content:

```
set printout default 2
set printout CellGlue none

onerror shutdown

#
check -strong setupFile
#
copy file:${setupFile} context:setupContext

#  import the variables into our $context.
#  don't overwrite already existing variables.
#
import context -c setupContext
#
#   Make sure we got what we need.
#
check -strong serviceLocatorHost serviceLocatorPort
check -strong statistics

create dmg.cells.services.RoutingManager  RoutingMgr

create dmg.cells.services.LocationManager lm \
   "${serviceLocatorHost} ${serviceLocatorPort}"


create diskCacheV111.services.PoolStatisticsV0 PoolStatistics  \
   "${statistics}  \
    -export \
#      -create  \
#      -htmlBase=${statistics}  \
    -domain=${thisFqHostname}"
```

The name of the file should be `statistics.batch`. Switch to the dCache jobs directory and run

```
[root] # ./initPackage.sh
```

Ignore possible error messages. All necessary links will be created.

Find a local disk area with sufficient space available to store the statistics data. The subdirectory should be empty and will be subsequently called (*/StatBase*).

Add the following line to the `context httpdSetup` section of the `config/httpd.batch` file.

```
set alias statistics directory /StatBase
```

Add the following line to the `config/dCacheSetup` file:

```
statistics=/StatBase
```

Make sure there is no other `statistics=..` entry.

Edit the file `docs/skins/home-skin-basic.html` : At two locations within this file, the statistics link is commented out. Undo this.

Important

The statistics link has to be called `href="/statistics/"`. Make sure the trailing `/` (slash) is present. This is not correcty done in the `docs/skins/home-skin-basic.html` file.

Finally restart the `httpd` and start the `statistics module`.

```
[root] # cd /opt/d-cache/jobs
[root] # ./httpd stop
[root] # ./httpd start
[root] # ./statistics start
```

Statistics is calculated once per hour at *HH*:55. The daily stuff is calculated at 23:55. Without manual intervention, it takes two midnights before all html statistics pages are available. There is a way to get this done after just one midnight. After the first midnight following the first startup of the statistics module, log into the `PoolStatistics` cell and run the following commands in the given sequence. The specified date has to be the Year/Month/Day of `today`.

```
create html Year Month Day
create html Year Month
create html Year
create html

Example (assuming today is April,11 2006)

create html 2006 04 11
create html 2006 04
create html 2006
create html
```

The statistics URL is

```
http://headnode:2288/statistics/
```

(Don’t forget the trailing slash)

Part IV. Reference

Table of Contents

Chapter 28. dCache Clients

The SRM Client Suite

An SRM URL has the form `srm://dmx.lbl.gov:6253//srm/DRM/srmv1?SFN=/tmp/try1` and the file URL looks like `file:////tmp/aaa`.

srmcp

srmcp — Copy a file from or to an SRM or between two SRMs.

Synopsis

`srmcp` [`option`...] {*`sourceUrl`*} {*`destUrl`*}

Arguments

sourceUrl The URL of the source file.

destUrl The URL of the destination file.

Options

`gss_expected_name` To enable the user to specify the gss expected name in the DN (Distinguished Name) of the srm server. The default value is `host`.

If the CN of host where srm server is running is `CN=srm/tam01.fnal.gov`, then `gss_expected_name` should be `srm`.

```
[user] $ srmcp --gss_expected_name=srm sourceUrl destinationUrl
```

`globus_tcp_port_range` To enable the user to specify a range of ports open for tcp connections as a pair of positive integers separated by "`:`", not set by default.

This takes care of compute nodes that are behind firewall.

`globus_tcp_port_range=40000:50000`

```
[user] $ srmcp --
globus_tcp_port_range=minVal:maxVal sourceUrl destinationUrl
```

`streams_num` To enable the user to specify the number of streams to be used for data transfer. If set to 1, then stream mode is used, otherwise extended block mode is used.

```
[user] $ srmcp --streams_num=1 sourceUrl destinationUrl
```

`server_mode` To enable the user to set the (gridftp) server mode for data transfer. Can be `active` or `passive`, `passive` by default.

This option will have effect only if transfer is performed in a stream mode (see `streams_num`)

```
[user] $ srmcp --streams_num=1 --
server_mode=active sourceUrl destinationUrl
```

Description

srmstage

srmstage — Request staging of a file.

Synopsis

`srmstage` [*`srmUrl`*...]

Arguments

srmUrl The URL of the file which should be staged.

Description

Provides an option to the user to stage files from HSM to dCache and not transfer them to the user right away. This case will be useful if files are not needed right away at user end, but its good to stage them to dcache for faster access later.

Chapter 29. dCache Cell Commands

This is the reference to all (important) cell commands in dCache. You should not use any command not documented here, unless you really know what you are doing. Commands not in this reference are used for debugging by the developers.

This chapter serves two purposes: The other parts of this book refer to it, whenever a command is mentioned. Secondly, an administrator may check here, if he wonders what a command does.

Common Cell Commands

pin

pin — Adds a comment to the pinboard.

Synopsis

`pin` { *`comment`* }

Arguments

comment A string which is added to the pinboard.

Description

info

info — Print info about the cell.

Synopsis

`info` [-a] [-l]

Arguments

-a Display more information.

-l Display long information.

Description

The info printed by **info** depends on the cell class.

dump pinboard

dump pinboard — Dump the full pinboard of the cell to a file.

Synopsis

`dump pinboard` { *`filename`* }

Arguments

filename The file the current content of the pinboard is stored in.

Description

show pinboard

show pinboard — Print a part of the pinboard of the cell to STDOUT.

Synopsis

`show pinboard` [*`lines`*]

Arguments

lines The number of lines which are displayed. Default: all.

Description

`PnfsManager` Commands

pnfsidof

pnfsidof — Print the `pnfs` id of a file given by its global path.

Synopsis

`pnfsidof` { *`globalPath`* }

Description

Print the `pnfs` id of a file given by its global path. The global path always starts with the “VirtualGlobalPath” as given by the “**info**”-command.

flags remove

flags remove — Remove a flag from a file.

Synopsis

`flags remove` { *`pnfsId`* } { *`key`* ...}

Arguments

pnfsId The `pnfs` id of the file of which a flag will be removed.

key flags which will be removed.

Description

flags ls

flags ls — List the flags of a file.

Synopsis

`flags ls` { *`pnfsId`* }

pnfsId The `pnfs` id of the file of which a flag will be listed.

Description

flags set

flags set — Set a flag for a file.

Synopsis

`flags set` { *`pnfsId`* } { *`key=value`* ...}

Arguments

pnfsId The `pnfs` id of the file of which flags will be set.

key The flag which will be set.

value The value to which the flag will be set.

Description

metadataof

metadataof — Print the meta-data of a file.

Synopsis

`metadataof` {[*`pnfsId`*] | [*`globalPath`*]} [-v] [-n] [-se]

Arguments

pnfsId The `pnfs` id of the file.

globalPath The global path of the file.

Description

pathfinder

pathfinder — Print the global or local path of a file from its PNFS id.

Synopsis

`pathfinder` { *`pnfsId`* } [[-global] | [-local]]

Arguments

pnfsId The `pnfs` Id of the file.

-global Print the global path of the file.

-local Print the local path of the file.

Description

set meta

set meta — Set the meta-data of a file.

Synopsis

`set meta` {[*`pnfsId`*] | [*`globalPath`*]} {*`uid`*} {*`gid`*} {*`perm`*} {*`levelInfo`*...}

Arguments

pnfsId The `pnfs` id of the file.

globalPath The global path oa the file.

uid The user id of the new owner of the file.

gid The new group id of the file.

perm The new file permitions.

levelInfo The new level information of the file.

Description

storageinfoof

storageinfoof — Print the storage info of a file.

Synopsis

`storageinfoof` {[*`pnfsId`*] | [*`globalPath`*]} [-v] [-n] [-se]

Arguments

pnfsId The `pnfs` id of the file.

globalPath The global path oa the file.

Description

cacheinfoof

cacheinfoof — Print the cache info of a file.

Synopsis

`cacheinfoof` {[*`pnfsId`*] | [*`globalPath`*]}

Arguments

pnfsId The `pnfs` id of the file.

globalPath The global path oa the file.

Description

Pool Commands

rep ls

rep ls — List the files currently in the repository of the pool.

Synopsis

`rep ls` [pnfsId...] | [-l= s | p | l | u | nc | e ...] [-s= k | m | g | t]

pnfsId The `pnfs` ID(s) for which the files in the repository will be listed.

-l List only the files with one of the following properties:

```
s      sticky files
p      precious files
l      locked files
u      files in use
nc     files which are not cached
e      files with an error condition
```

-s Unit, the filesize is shown:

```
k      data amount in KBytes
m      data amount in MBytes
g      data amount in GBytes
t      data amount in TBytes
```

Description

st set max active

st set max active — Set the maximum number of active store transfers.

Synopsis

`st set max active` {*`maxActiveStoreTransfers`*}

maxActiveStoreTransfers The maximum number of active store transfers.

Description

Any further requests will be queued. This value will also be used by the *cost module* for calculating the *performance cost*.

rh set max active

rh set max active — Set the maximum number of active restore transfers.

Synopsis

`rh set max active` {*`maxActiveRetoreTransfers`*}

maxActiveRetoreTransfers — The maximum number of active restore transfers.

Description

Any further requests will be queued. This value will also be used by the *cost module* for calculating the *performance cost*.

mover set max active

mover set max active — Set the maximum number of active client transfers.

Synopsis

`mover set max active` {*`maxActiveClientTransfers`*} [`-queue=`*`moverQueueName`*]

maxActiveClientTransfers — The maximum number of active client transfers.

moverQueueName — The mover queue for which the maximum number of active transfers should be set. If this is not specified, the default queue is assumed, in order to be compatible with previous versions which did not support multiple mover queues (before version 1.6.6).

Description

Any further requests will be queued. This value will also be used by the *cost module* for calculating the *performance cost*.

p2p set max active

p2p set max active — Set the maximum number of active pool-to-pool server transfers.

Synopsis

`p2p set max active` {*`maxActiveP2PTransfers`*}

maxActiveP2PTransfers — The maximum number of active pool-to-pool server transfers.

Description

Any further requests will be queued. This value will also be used by the *cost module* for calculating the *performance cost*.

pp set max active

pp set max active — Set the value used for scaling the performance cost of pool-to-pool client transfers analogous to the other `set max active`-commands.

Synopsis

`pp set max active` {*`maxActivePPTransfers`*}

maxActivePPTransfers The new scaling value for the cost calculation.

Description

All pool-to-pool client requests will be performed immediately in order to avoid deadlocks. This value will only used by the *cost module* for calculating the *performance cost*.

set gap

set gap — Set the gap parameter - the size of free space below which it will be assumed that the pool is full within the cost calculations.

Synopsis

`set gap` {*`gapPara`*}

gapPara The size of free space below which it will be assumed that the pool is full. Default is 4GB.

Description

The gap parameter is used within the space cost calculation scheme described in the section called “The Space Cost”. It specifies the size of free space below which it will be assumed that the pool is full and consequently the least recently used file has to be removed if a new file has to be stored on the pool. If, on the other hand, the free space is greater than gapPara, it will be expensive to store a file on the pool which exceeds the free space.

set breakeven

set breakeven — Set the breakeven parameter - used within the cost calculations.

Synopsis

`set breakeven` {*`breakevenPara`*}

breakevenPara The breakeven parameter has to be a positive number smaller than 1.0. It specifies the impact of the age of the *least recently used file* on space cost. It the LRU file is one

week old, the space cost will be equal to `(1 + breakeven)`. Note that this will not be true, if the breakeven parameter has been set to a value greater or equal to 1.

Description

The breakeven parameter is used within the space cost calculation scheme described in the section called “The Space Cost”.

mover ls

mover ls — List the active and waiting client transfer requests.

Synopsis

`mover ls` [{-queue} | {-queue=*queueName*}]

queueName The name of the mover queue for which the transfers should be listed.

Description

Without parameter all transfers are listed. With `-queue` all requests sorted according to the mover queue are listed. If a queue is explicitly specified, only transfers in that mover queue are listed.

migration cache

migration cache — Caches replicas on other pools.

SYNOPSIS

`migration cache` [*options*] *target*…

DESCRIPTION

Caches replicas on other pools. Similar to **migration copy**, but with different defaults. See **migration copy** for a description of all options. Equivalent to: **migration copy** -smode=same -tmode=cached

migration cancel

migration cancel — Cancels a migration job

SYNOPSIS

`migration cancel` [-force] job

DESCRIPTION

Cancels the given migration job. By default ongoing transfers are allowed to finish gracefully.

migration clear

migration clear — Removes completed migration jobs.

SYNOPSIS

```
migration clear
```

DESCRIPTION

Removes completed migration jobs. For reference, information about migration jobs are kept until explicitly cleared.

migration concurrency

migration concurrency — Adjusts the concurrency of a job.

SYNOPSIS

`migration concurrency` *`job`* *`n`*

DESCRIPTION

Sets the concurrency of *`job`* to *`n`*.

migration copy

migration copy — Copies files to other pools.

SYNOPSIS

`migration copy` [*`options`*] *`target`*…

DESCRIPTION

Copies files to other pools. Unless filter options are specified, all files on the source pool are copied.

The operation is idempotent, that is, it can safely be repeated without creating extra copies of the files. If the replica exists on any of the target pools, then it is not copied again. If the target pool with the existing replica fails to respond, then the operation is retried indefinitely, unless the job is marked as eager.

Both the state of the local replica and that of the target replica can be specified. If the target replica already exists, the state is updated to be at least as strong as the specified target state, that is, the lifetime of sticky bits is extended, but never reduced, and cached can be changed to precious, but never the opposite.

Jobs can be marked permanent. Permanent jobs never terminate and are stored in the pool setup file with the 'save' command. Permanent jobs watch the repository for state changes and copy any replicas that match

the selection criteria, even replicas added after the job was created. Notice that any state change will cause a replica to be reconsidered and enqueued if it matches the selection criteria - also replicas that have been copied before.

Options

-state=cached|precious
: Only copy replicas in the given state.

-sticky[=*owner*[,*owner*...]]
: Only copy sticky replicas. Can optionally be limited to the list of owners. A sticky flag for each owner must be present for the replica to be selected.

-storage=*class*
: Only copy replicas with the given storage class.

-pnfsid=*pnfsid*[,*pnfsid*] ...
: Only copy replicas with one of the given PNFS IDs.

-accessed=*n*|[*n*]..[*m*]
: Only copy replicas accessed n seconds ago, or accessed within the given, possibly open-ended, interval. E.g. -accessed=0..60 matches files accessed within the last minute; -accesed=60.. matches files accessed one minute or more ago.

-size=*n*|[*n*]..[*m*]
: Only copy replicas with size n, or a size within the given, possibly open-ended, interval.

-smode=same|cached|precious|removable|delete[+*owner*[(*lifetime*)] ...]
: Update the local replica to the given mode after transfer:

  same — does not change the local state (this is the default).

  cached — marks it cached.

  precious — marks it precious.

  removable — marks it cached and strips all existing sticky flags exluding pins.

  delete — deletes the replica unless it is pinned.

  An optional list of sticky flags can be specified. The lifetime is in seconds. A lifetime of 0 causes the flag to immediately expire. Notice that existing sticky flags of the same owner are overwritten.

-tmode=same|cached|precious[+*owner*[(*lifetime*)]...]
: Set the mode of the target replica:

  same — applies the state and sticky bits excluding pins of the local replica (this is the default).

  cached — marks it cached.

  precious — marks it precious.

  An optional list of sticky flags can be specified. The lifetime is in seconds.

-pins=move|keep
: Controls how sticky flags owned by the pin manager is handled:

move Ask pin manager to move pins to the target pool.

keep Keep pin on the source pool.

-select=proportional|best|random — Determines how a pool is selected from the set of target pools:

proportional selects a pool with a probability inversely proportional to the cost of the pool.

best selects the pool with the lowest cost.

random selects a pool randomly.
The default is proportional.

-target=pool|pgroup|link — Determines the interpretation of the target names. The default is 'pool'.

-refresh=*time* — Specifies the period in seconds of when target pool information is queried from the pool manager. The default is 300 seconds.

-exclude=*pool*[,*pool*...] — Exclude target pools.

-concurrency=*concurrency* — Specifies how many concurrent transfers to perform. Defaults to 1.

-eager — Copy replicas rather than retrying when pools with existing replicas fail to respond.

-permanent — Mark job as permanent.

migration info

migration info — Shows detailed information about a migration job.

SYNOPSIS

`migration info` *job*

DESCRIPTION

Shows detailed information about a migration job. Possible job states are:

INITIALIZING Initial scan of repository

RUNNING Job runs (schedules new tasks)

SLEEPING A task failed; no tasks are scheduled for 10 seconds

SUSPENDED Job suspended by user; no tasks are scheduled

CANCELLING Job cancelled by user; waiting for tasks to stop

CANCELLED	Job cancelled by user; no tasks are running
FINISHED	Job completed

Job tasks may be in any of the following states:

Queued	Queued for execution
GettingLocations	Querying PnfsManager for file locations
UpdatingExistingFile	Updating the state of existing target file
CancellingUpdate	Task cancelled, waiting for update to complete
InitiatingCopy	Request send to target, waiting for confirmation
Copying	Waiting for target to complete the transfer
Pinging	Ping send to target, waiting for reply
NoResponse	Cell connection to target lost
Waiting	Waiting for final confirmation from target
MovingPin	Waiting for pin manager to move pin
Cancelling	Attempting to cancel transfer
Cancelled	Task cancelled, file was not copied
Failed	The task failed
Done	The task completed successfully

migration ls

migration ls — Lists all migration jobs.

SYNOPSIS

```
migration ls
```

DESCRIPTION

Lists all migration jobs.

migration move

migration move — Moves replicas to other pools.

SYNOPSIS

`migration move` [*`options`*] *`target`*...

DESCRIPTION

Moves replicas to other pools. The source replica is deleted. Caches replicas on other pools. Similar to **migration copy**, but with different defaults. Accepts the same options as **migration copy**. Equivalent to: **migration copy** -smode=delete -tmode=same -pins=move

migration suspend

migration suspend — Suspends a migration job.

SYNOPSIS

`migration suspend` job

DESCRIPTION

Suspends a migration job. A suspended job finishes ongoing transfers, but is does not start any new transfer.

migration resume

migration resume — Resumes a suspended migration job.

SYNOPSIS

`migration resume` job

DESCRIPTION

Resumes a suspended migration job.

`PoolManager` Commands

rc ls

rc ls — List the requests currently handled by the `PoolManager`

Synopsis

`rc ls` [*`regularExpression`*] [`-w`]

Description

Lists all requests currently handled by the pool manager. With the option `-w` only the requests currently waiting for a response are listed. Only requests satisfying the regular expression are shown.

cm ls

cm ls — List information about the pools in the *cost module* cache.

Synopsis

`cm ls` [-r] [-d] [-s] [*`fileSize`*]

-r Also list the tags, the *space cost*, and *performance cost* as calculated by the cost module for a file of size *`fileSize`* (or zero)

-d Also list the *space cost* and *performance cost* as calculated by the cost module for a file of size *`fileSize`* (or zero)

-t Also list the time since the last update of the cached information in milliseconds.

Description

A typical output reads

```
(PoolManager) admin > cm ls -r -d -t 12312434442
poolName1={R={a=0;m=2;q=0};S={a=0;m=2;q=0};M={a=0;m=100;q=0};PS={a=0;m=20;q=0};PC={a=0;m=20;q=0};
    (...line continues...)  SP={t=2147483648;f=924711076;p=1222772572;r=0;lru=0;{g=20000000;b=0.5}}}
poolName1={Tag={{hostname=hostname}};size=543543543;SC=1.7633947200606475;CC=0.0;}
poolName1=3180
poolName2={R={a=0;m=2;q=0};S={a=0;m=2;q=0};M={a=0;m=100;q=0};PS={a=0;m=20;q=0};PC={a=0;m=20;q=0};
    (...line continues...)  SP={t=2147483648;f=2147483648;p=0;r=0;lru=0;{g=4294967296;b=250.0}}}
poolName2={Tag={{hostname=hostname}};size=543543543;SC=0.0030372862312942743;CC=0.0;}
poolName2=3157
```

set pool decision

set pool decision — Set the factors for the calculation of the total costs of the pools.

Synopsis

`set pool decision` [-spacecostfactor=*`scf`*] [-cpucostfactor=*`ccf`*] [-costcut=*`cc`*]

scf The factor (strength) with which the *space cost* will be included in the *total cost*.

ccf The factor (strength) with which the *performance cost* will be included in the *total cost*.

cc Deprecated since version 5 of the pool manager.

Description

Chapter 30. dCache Developers Corner

This chapter provides useful informatiom for dCache developers and is included as reference material and theory of operation.

The StorageInfoQuotaObserver cell

The StorageInfoQuotaObserver keeps track on spaces for all attached pools. The space granularity is based on the StorageInfo. It records precious, total, pinned, free and removable spaces of currently available pools. Pools, not active are not counted. Spaces may be queried by pool, storageinfo or link. For link queries, additional, link specific information is provided for convenience.

Calling Sequence

```
#
define context QuotaManagerSetup  endDefine
   set pool query interval 180
   set pool query steps     20
   set pool query break    200
   set poolmanager query interval 350
   set pool validity timeout 400
endDefine
#
create diskCacheV111.services.space.StorageInfoQuotaObserver QuotaManager \
              "default -export"
#
```

Parameter setter commands

These commands allow to customize the behaviour of the StorageInfoQuotaObserver. They many determine how often information is updated and how aggressive the cells queries other services for updates. The meaning of the `set pool/poolmanager query interval` is obvious. Because of the fact, that the number of pools to query can be rather large, the cell allows to send the space update queries in junks with some time inbetween. The junk size is set by `set pool query steps` and the break between sending junks by `set pool query break`. If no pool information arrived within the `set pool validity timeout` the corresponding pool is declared OFFLINE and the spaces are no longer counted.

Table 30.1. Parameter setting reference

Command	Argument Type	Argument Unit	Meaning
set pool query interval	Time	Seconds	Time interval between pool space queries
set poolmanager query interval	Time	Seconds	Time interval between pool manager pool/link queries
set pool query break	Time	Milli-seconds	Time interval between pool query 'steps'
set pool query steps	Counter	None	Number of space queries between 'break'
set pool validity timeout	Time	Seconds	If if pool info arrived within this time, the pool is declared OFFLINE

Information query commands

- **show pool [*poolName*]**
- **show link [*-a*]** Lists spaces per link. The *-a* option provides additional information, eg. the storage classes and pools assigned to the particular link.
- **show sci** Lists spaces per storage element.

Messages

This cells currently replies on the following cell messages. The different sections are all relative to `diskCacheV111.vehicles.`

`PoolMgrGetPoolLinks`

The StorageInfoQuotaCell provides a list of `PoolLinkInfo` structures, one per known link, on arrival of the message. Each `PoolLinkInfo` is filled with the name of the link, the list of storage classes, this link is associtated to, and the totally available space, left in this link. OFFLINE pools are not counted.

`QuotaMgrCheckQuotaMessage`

StorageInfoQuotaCell provides the soft and hard quota defined for the specified `StorageClass` together with the space used.

Chapter 31. dCache default port values

Table 31.1.

Port number	Description	Component
32768 and 32768	is used by the NFS layer within dCache which is based upon rpc. This service is essential for rpc.	NFS
1939 and 33808	is used by portmapper which is also involved in the rpc dependencies of dCache.	portmap
34075	is for postmaster listening to requests for the PostgreSQL database for dCache database functionality.	Outbound for `SRM`, PnfsDomain, dCacheDomain and doors; inbound for PostgreSQL server.
33823	is used for internal dCache communication.	By default: outbound for all components, inbound for dCache domain.
8443	is the `SRM` port. See Chapter 14, *dCache Storage Resource Manager*	Inbound for `SRM`
2288	is used by the web interface to dCache.	Inbound for httpdDomain
22223	is used for the dCache admin interface. See the section called “The Admin Interface”	Inbound for adminDomain
22125	is used for the dCache `dCap` protocol.	Inbound for `dCap` door
22128	is used for the dCache `GSIdCap` .	Inbound for `GSIdCap` door

Chapter 32. Glossary

The following terms are used in dCache.

tertiary storage system
: A mass storage system which stores data and is connected to the dCache system. Each dCache pool will write files to it as soon as they have been completely written to the pool (if the pool is not configured as a *LFS*). The tertiary storage system is not part of dCache. However, it is possible to connect any mass storage system as tertiary storage system to dCache via a simple interface.

tape backend
: A *tertiary storage system* which stores data on magnetic tapes.

Hierarchical Storage Manager (HSM)
: See tertiary storage system.

HSM type
: The type of HSM which is connected to dCache as a *tertiary storage system*. The choice of the HSM type influences the communication between dCache and the HSM. Currently there are `osm` and `enstore`. `osm` is used for most HSMs (TSM, HPSS, ...).

Large File Store (LFS)
: A Large File Store is the name for a dCache instance that is acting as a filesystem independent to, or in cooperation with, an *HSM* system. When dCache is acting as an LFS, files may be stored and later read without involving any HSM system.

  Whether a dCache instance provides an LFS depends on whether there are *pools* configured to do so. The `LFS` option, specified for each pool within the *`poollist` file*, describes how that pool should behave. This option can take three possible values:

  `none`
  : the pool does not contribute to any LFS capacity. All newly written files are regarded precious and sent to the HSM backend.

  `precious`
  : Newly create files are regarded as precious but are not scheduled for the HSM store procedure. Consequently, these file will only disappear from the pool when deleted in the *namespace*.

  `volatile` (or `transient`)
  : Newly create files are regarded cached and are not scheduled for the HSM store procedure. Though they will never be stored on tape, these file are part of the aging procedure and will be removed as soon as new space is needed.

Note

The `volatile` lfs mode is deprecated and should not be used.

to store — Copying a file from a dCache pool to the *tertiary storage system*.

to restore — Copying a file from the *tertiary storage system* to one of the dCache pools.

to stage — See to restore.

transfer — Any kind of transfer performed by a dCache pool. There are *store*, *restore*, pool to pool (client and server), read, and write transfers. The latter two are client transfers.
See Also mover.

mover — The process/thread within a *pool* which performs a *transfer*. Each pool has a limited number of movers that may be active at any time; if this limit is reached then further requests for data are queued.

In many protocols, end clients connect to a mover to transfer file contents. To support this, movers must speak the protocol the end client is using.
See Also transfer.

The `dCacheSetup` File — This is the primary configuration file of a dCache host. It is located at `$dcache_home/config/dCacheSetup` (typically `/opt/d-cache/config/dCacheSetup`). Each *domain* uses the file `config/`*`domainName`*`Setup` which is in fact a symbolic link to `config/dCacheSetup`. The `config/dCacheSetup` file might even be the same across the hosts of a dCache instance.

Primary Network Interface

`poollist` File — The poollist files are a collection of files in the `/opt/d-cache/config` directory. Each `poollist` file describes the set of *pools* that should be available for a given node. These files have a filename like *`hostname`*`.poollist`, where *`hostname`* is the simple hostname of the node the pools are to run on.

The file consists of one or more lines, with each line describing a pool.

Location Manager — The location manager is a *cell* that instructs a newly started *domain*s to which domain they should connect. This allows domains to form arbitrary network topologies; although, by default, a dCache instance will form a star topology with the `dCacheDomain` domain at the centre.

Cell — A cell is a collection of Java threads that provide a discrete and simple service within dCache. Each cell is hosted within a *domain*.

	Cells have an address derived from concatenating their name, the `@` symbol and their containing domain name.
Domain	A domain is a collection of one or more *cell*s that provide a set of related services within a dCache instance. Each domain requires its own Java Virtual Machine. A typical domain might provide external connectivity (i.e., a *door*) or manage the *pools* hosted on a machine. Each domain has at least one cell, called the `System` cell and many tunnel cells for communicating with other Domains. To provide a useful service, a domain will contain other cells that provide specific behaviour.
Door	Door is the generic name for special *cell*s that provides the first point of access for end clients to communicate with a dCache instance. There are different door implementations (e.g., `GSIdCap` door and `GridFTP` door), allowing a dCache instance to support multiple communication protocols. A door will (typically) bind to a well-known port number depending on the protocol the door supports. This allows for only a single door instance per machine for each protocol. A door will typically identify which *pool* will satisfy the end user's operation and redirect the client to the corresponding pool. In some cases this is not possible; for example, some protocols (such as `GridFTP` version 1) do not allow servers to redirect end-clients, in other cases pool servers may be behind a firewall, so preventing direct access. When direct end-client access is not possible, the door may act as a data proxy, streaming data to the client. By default, each door is hosted in a dedicated *domain*. This allows easy control of whether a protocol is supported from a particular machine.
Java Virtual Machine (JVM)	Java programs are typically compiled into a binary form called Java byte-code. Byte-code is comparable to the format that computers understand native; however, no mainstream processor understands Java byte-code. Instead compiled Java programs typically require a translation layer for them to run. This translation layer is called a Java Virtual Machine (JVM). It is a standardised execution environment that Java programs may run within. A JVM is typically represented as a process within the host computer.
Well Known Cell	A well-known cell is a *cell* that registers itself centrally. Within the admin interface, a well-known cell may be referred to by just its cell name.
Pinboard	The pinboard is a collection of messages describing events within dCache and is similar to a log file. Each *cell* will (typically) have its own pinboard.
Breakeven Parameter	
Secondary Network Interface	

least recently used (LRU) File

Default Mover Queue

Namespace

The namespace is a core component of dCache. It maps each stored file to a unique identification number and allows storing of metadata against either files or directories.

dCache supports two (independent) namespace implementations: *pnfs* and *Chimera*.

pnfs filesystem

pnfs is a filesystem that uses a database to store all information, including the contents of files. This filesystem is made available via NFS, so authorised hosts can mount pnfs and use it like any other file system.

dCache may use pnfs as its *namespace*. Although it is possible to store file contents in pnfs, dCache does not do this. Instead dCache stores the file data on one (or more) *pools*.

pnfs includes some unique additional properties. These include *dot commands*, *pnfs IDs*, *levels*, *directory tags* and *wormholes*.

pnfs dot command

To configure and access some of the special features of the *pnfs filesystem*, special files may be read, written to or created. These files all start with a dot (or period) and have one or more parameters after, each parameter is contained within a set of parentheses; for example, the file .(tag)(*foo*) is the pnfs dot command for reading or writing the *foo directory tag* value.

Care must be taken when accessing a dot command from a shell. Shells will often expand parentheses so the filename must be protected against this; for example, by quoting the filename or by escaping the parentheses.

pnfs level

In *pnfs*, each file can have up to eight independent contents; these file-contents, called levels, may be accessed independently. dCache will store some file metadata in levels 1 and 2, but dCache will not store any file data in pnfs.

pnfs directory tag

pnfs includes the concept of tags. A tag is a keyword-value pair associated with a directory. Subdirectories inherit tags from their parent directory. New values may be assigned, but tags cannot be removed. The *dot command* .(tag)(*foo*) may be used to read or write tag *foo*'s value. The dot command .(tags)() may be read for a list of all tags in that file's subdirectory.

More details on directory tags are given in the section called "Directory Tags".

pnfs ID

Each component (file, directory, etc) in a pnfs instance has a unique ID: a 24-digit hexadecimal number. This unique ID is used in dCache to refer

to files without knowing the component's name or in which directory the component is located.

More details on `pnfs` IDs are given in the section called "pnfsIDs".

Pool to Pool Transfer

A pool-to-pool transfer is one where a file is transferred from one dCache *pool* to another. This is typically done to satisfy a read request, either as a load-balancing technique or because the file is not available on pools that the end-user has access.

Storage Class

`batch` File

A batch file describes which *cell*s in a *domain* are to be started and with what options. They typically have filenames from combining the name of a domain with `.batch`; for example, the `dCacheDomain` domain has a corresponding batch file `dCache.batch`

Although the cells in a domain may be configured by altering the corresponding batch file, most typical changes can be altered by editing the `dCacheConfig` file and this is the preferred method.

Context

Wormhole

A wormhole is a feature of the *`pnfs` filesystem*. A wormhole is a file that is accessible in all directories; however, the file is not returned when scanning a directory(e.g., using the **ls** command).

More details on wormholes are given in the section called "Global Configuration with *Wormholes*".

Chimera

Chimera is a *namespace* implementation that is similar to *`pnfs`* but provides better integration with a relational database. Because of this, it allows additional functionality to be added, so some dCache features require a chimera namespace.

Many *`pnfs`* features are available in Chimera, including *levels*, *directory tags* and many of the *dot commands*.

Chimera ID

A *Chimera* ID is a 36 hexadecimal digit that uniquely defines a file or directory. It's equivalent to a *`pnfs` ID*.

Replica

It is possible that dCache will choose to make a file accessible from more than one *pool* using a *pool-to-pool* copy. If this happens, then each copy of the file is a replica.

A file is independent of which pool is storing the data whereas a replica is uniquely specified by the `pnfs` ID *and* the pool name it is stored on.

Precious Replica

A precious replica is a *replica* that should be stored on tape.

Cached Replica

A cached replica is a *replica* that should not be stored on tape.

Replica Manager

The replica manager keeps track of the number of *replicas* of each file within a certain subset of pools and makes sure this number is always within a specified range. This way, the system makes sure that enough versions of each file are present and accessible at all times. This is especially useful to ensure resilience of the dCache system, even if the hardware is not reliable. The replica manager cannot be used when the system is connected to a *tertiary storage system*. The activation and configuration of the replica manager is described in Chapter 6, *Resilience with the Replica Manager*.

Storage Resource Manager (SRM)

An SRM provides a standardised webservice interface for managing a storage resource (e.g. a dCache instance). It is possible to reserve space, initiate file storage or retrieve, and replicate files to another SRM. The actual transfer of data is not done via the SRM itself but via any protocol supported by both parties of the transfer. Authentication and authorisation is done with the grid security infrastructure. dCache comes with an implementation of an SRM which can turn any dCache instance into a grid storage element.

`pnfs` Companion

The `pnfs` companion is a (database) table that stores dCache specific information; specifically, on which pools a file may be found. dCache can operate without a companion and will store file location information within a *level*.

Storing replica location information in the companion database greatly improves the performance of dCache as the location information is often queried by the *pool manager*.

Although a companion database may be used with *Chimera*, doing so provides no performance improvements and is not recommended.

Billing/Accounting

Accounting information is either stored in a text file or in a PostgreSQL database by the `billing` *cell* usually started in the `httpDomain` *domain*. This is described in Chapter 24, *Accounting*.

Pool Manager

The pool manager is the *cell* running in the `dCacheDomain` *domain*. It is a central component of a dCache instance and decides which pool is used for an incoming request.

Cost Module

The cost module is a Java class responsible for combining the different types of cost associated with a particular operation; for example, if a file is to be stored, the cost module will combine the storage costs and CPU costs for each candidate target pool. The pool manager will choose the candidate pool with the least combined cost.

Pool Selection Unit

The pool selection unit is a Java class responsible for determining the set of candidate pools for a specific transaction. A detailed account of its configuration and behaviour is given in the section called “The Pool Selection Mechanism”.

Pin Manager

The pin manager is a *cell* by default running in the `utility` *domain*. It is a central service that can “pin” files to a pool for a certain time. It is used by the `SRM` to satisfy prestage requests.

Space Manager

The (SRM) Space Manager is a *cell* by default running in the `srm` *domain*. It is a central service that records reserved space on pools. A space reservation may be either for a specific duration or never expires. The Space Manager is used by the `SRM` to satisfy space reservation requests.

Pool

A pool is a *cell* responsible for storing retrieved files and for providing access to that data. Data access is supported via *mover*s. A machine may have multiple pools, perhaps due to that machine's storage being split over multiple partitions.

A pool must have a unique name and all pool cells on a particular machine are hosted in a *domain* that derives its name from the host machine's name.

The list of directories that are to store pool data are found in the *`poollist` File*, which is located on the pool node.

sweeper

A sweeper is an activity located on a *pool*. It is responsible for deleting files on the pool that have been marked for removal. Files can be marked for removal because their corresponding *namespace* entry has been deleted or because the local file is a *cache copy* and more disk space is needed.

HSM sweeper

The HSM sweeper, if enabled, is a component that is responsible for removing files from the *HSM* when the corresponding *namespace* entry has been removed.

cost

The pool manager determines the pool used for storing a file by calculating a cost value for each available pool. The pool with the lowest cost is used. The costs are calculated by the cost module as described in . The total cost is a linear combination of the I.e.,

where `ccf` and `scf` are configurable with the command **set pool decision**.

performance cost

See Also gl-cost.

space cost

See Also gl-cost.