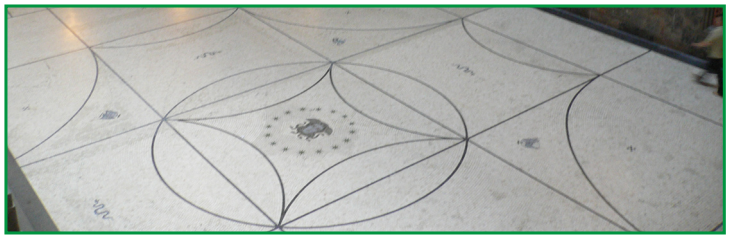
LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

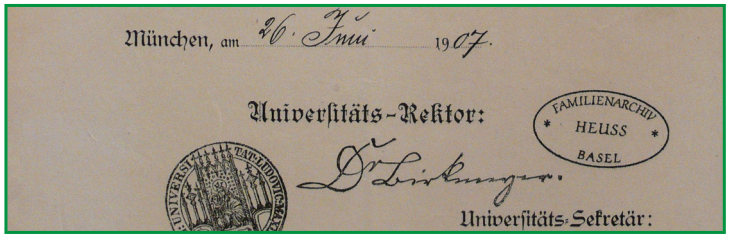
GRIDKA SCHOOL 2014
KARLSRUHER INSTITUT FÜR TECHNOLOGIE
KARLSRUHE

September 4, 2014

dCache Introduction Course

Christoph Anton Mitterer
christoph.anton.mitterer@lmu.de





I. Introduction To dCache

Storage Management Systems

dCache is a so called *storage management system*. These systems manage files like "normal" filesystems (for example btrfs, ext4 or XFS) do, but usually on a much higher logical level.

Storage management systems are characterised by the following main attributes:

- Manage much higher amounts of data than normal filesystems usually do.
- Integrate many storage media that can be of different type into one system.
- Utilise **H**ierarchical **S**torage **M**anagement (for example hard disk ⇄ tape).
- Provide mechanisms to automatically increase the performance and balance the load (for example "auto-replication").
- Provide means for resilience and high availability.
- Provide advanced control systems to manage the data as well as the data flow.
- May provide interfaces to several access-protocols (for example POSIX or SRM).

As a component of a computing centre or a grid, an instance of a storage management system (including the actual data) is often referred to as "***S**torage **E**lement*" (in contrast to the "***C**ompute **E**lement*").

Overview And History

- dCache is a highly sophisticated storage management system written mostly in Java.
- It is being actively developed at DESY since 2003.
  Major contributions come from FNAL and NDGF.
- It uses standard open source technologies in many places.
- It is released under the GNU Affero General Public License version 3
  and some parts under the GNU Lesser General Public License version 3.
- It is one of the main storage management systems used within the European Middleware Initiative and the LHC Computing Grid and it is used in different other places.
- Support is available through the developers, dedicated support groups and mailing lists.

Major release-branches of dCache include:

- 2.6: Old stable branch.
- 2.10: Current stable branch.
- git: Development branch.

Functionalities And Features

The main functionalities and features of dCache include:

- Management of enormous amounts of data (no hard limit, tested in the petabyte-range).
- **H**ierarchical **S**torage **M**anagement
- Scalability, reasonable performance, modular design and portability.
- Highly configurable and customisable, thus allowing very mighty setups.
- Some support for the usage of different database management systems.
- Use of “normal” filesystems (for example btrfs, ext4 or XFS) to store the actual data on the storage-nodes.
- X.509-based authentication through TLS/SSL or **G**rid **S**ecurity **I**nfrastructure and powerful authorisation systems with support for access control lists.
- Kerberos-based authentication.
- Firewall-friendly by utilising the concept of door-nodes.
- Access interfaces for NFS 4.1 (pNFS), HTTP and WebDAV, gsiFTP (“GridFTP”), rootd (“xrootd”) and SRM (version 1 to 2.2) in addition to its legacy native protocols DCAP and gsiDCAP.
- Scriptable administration interface with terminal-based and graphical front-ends.

Functionalities And Features

- Detailed logging and debugging as well as accounting and statistics.
- XML information system with detailed live information about the cluster.
- Web-interface with live summaries of the most important information and some administration functionality.
- Checksumming of data.
- Easy migration and management of data via the “migration module”.
- Resilience and high availability can be implemented on different ways by having multiple replicas of the same files.
- Powerful cost calculation systems that allows to control the data flow (from and to pools, between pools and also between pools and tape).
- Load balancing and performance tuning by “hot pool replication” (via cost calculation and replicas created by pool-to-pool-transfers).
- Garbage collection of replicas, depending on their flags, age, et cetera.
- Space management and support for space tokens.
- *Et cetera.*

This course does not cover the HSM-capabilities (“writing to tape”) of dCache.

Future And Planned Developments

dCache is under active development with the current focus on the following topics:

- Improving or rewriting parts of the existing code, especially with respect to functionality, performance and compatibility with other clients.
- Cloud-related technologies.
- Eventually perhaps striping of files.
- Further deploying standardised technologies in many areas of dCache.
- *Et cetera.*

Organisations That Are Using dCache

Some organisations that are using dCache include:

RWTH RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

Fraunhofer

NORDUnet Nordic Infrastructure for Research & Education

Open Science Grid

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

MAX-PLANCK-GESELLSCHAFT

EUROPEAN MIDDLEWARE INITIATIVE

lrz

DESY

KIT Karlsruhe Institute of Technology

WLCG Worldwide LHC Computing Grid

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Contact Addresses

You should note the following contact addresses:

- dCache Website
  `http://www.dcache.org/`
  (includes the “dCache-Book” and a dCache-Wiki)
- dCache Support
  `support@dcache.org`
- dCache User Mailing List
  `user-forum@dcache.org`
- Mass Storage Support Group of the Helmholtz-Gemeinschaft Deutscher Forschungszentren

  through the mailing list:
  `german-support@dcache.org`

  through the **G**lobal **G**rid **U**ser **S**upport:
  `http://ggus.eu/`

dCache's Structure

dCache is a modularised system that consists of many different components, where each component fulfils a dedicated task. Depending on the specific task, a component may occur exactly once or multiple times per cluster.

A dCache cluster is structured by the following two classes of objects:

- Domains
  A domain consists of one or more cells, grouping them (typically logically) together.
  Each domain corresponds to exactly one Java VM and is thus bound to exactly one node of the cluster.
  Nodes can generally run multiple domains.
- Service
  A service is the actual entity that fulfils a specific task and thus corresponds roughly to one of the components mentioned above.
  Depending on the service-type there may be multiple instances per cluster, possibly each running with a different set of settings.

dCache's Structure

Technically, there is actually a third class of objects:

- Cells
  The services mentioned before are provided by one or more cells (possibly from different domains), grouping them semantically together.
  Cells may be mandatory or optional for the service's functionality.
  Each cell belongs to exactly one domain.

Service-Classes

The different types of services are divided in the following classes:

- Core-services
  These implement basic or common tasks for the whole cluster, for example inter-cell-communication, access to the file hierarchy provider or accounting.
  Many types of core-services can exist only once per cluster.
- Door-services (“doors”)
  These serve as gateways to the actual data and provide accessing clients with an interface speaking a given access-protocol (Depending on the protocol, its configuration and the client’s request, the actual data flows either directly between the client and the pool or indirectly via the door.).
  The different types of doors can exist multiple times per cluster, typically placed on different nodes (so called “door-nodes”).
- Pool-services (“pools”)
  These manage the actual data on the storage-nodes (so called “pool-nodes”) and can even communicate directly with the accessing clients.
  Of course, there may be many pools per cluster, but even per node.

Important Core-Service And -Cell-Types

The following describes some more important core-service and -cell-types:

- core cells in every domain
  The `RoutingMgr`- and `lm`-cells are used with dCache's native cell-messaging (and to some extent also with JMS).
  They maintain the routes to domains and cells known to the specific "local" domain as well as network communication information.
  In the default configuration, `dCacheDomain`'s `RoutingMgr`-cell has knowledge on all routes within the cluster and its `lm`-cell acts as a server to all the other `lm`-(client-)cells.
  The `System`-cells can be used to "explore" the respective domain itself.
- `poolmanager`-service
  Manages and configures the pools as well as the data flow ("pool selection") from and to pools, between pools and also between pools and tape.

Important Core-Service And -Cell-Types

- `pnfsmanager`-service (called so for historical reasons)
  The general interface to the file hierarchy (and thus to its database), providing an interface to the other cells. It allows access to all the file meta data, including pathnames, file properties, ACLs and PNFS-IDs.
- `admin`-servce
  dCache's administration interface.
- `gplazma`-service
  Provides functionalities for access control, that are used by doors in order to implement their access control system.
  See chapter V.
- `spacemanager`-service
  Handles space management and space reservation.
- `pinmanager`-service
  Handles the pinning of files (mainly used with HSM-systems and in combination with SRM).

Important Core-Service And -Cell-Types

- `httpd`-service
  The web interface and web server.
- `info`-service
  The information system, which provides all kinds of live information to other services.
- `billing`-service
  The accounting system.
- `statistics`-service
  The statistics collection system.
- `loginbroker`-service
  Maintains a list of doors as well as their protocol, version, fully qualified domain name, port and some other data.
- `dir`-service
  Provides the listing of directories with the legacy protocols DCAP and gsiDCAP.

Access-Protocols And Door-Types

dCache provides accessing clients with interfaces (so called “doors”) for a number of protocols. A door runs usually (but not necessarily) in one domain and all types of doors can have in principle multiple instances per cluster.

Currently, there are door-types for the following protocols:

- NFS 3
  NFS version 3 is a network filesystem, which allows rather limited access to the file hierarchy itself and the file-meta-data.
- NFS 4.1 (pNFS)
  NFS version 4.1 is a network filesystem defining parallel NFS, which allows “full” (and “direct”) access to the files.
- WebDAV
  The WebDAV protocol, which is a superset of HTTP.
- FTP
  Plain, authenticated as well as Kerberos- and GSI-secured (“GridFTP”) versions of the the **F**ile **T**ransfer **P**rotocol.

Access-Protocols And Door-Types

- DCAP
  Plain, authenticated as well as Kerberos- and GSI-secured versions of the **dC**ache **A**ccess **P**rotocol, dCache's legacy native access protocol.
- rootd ("xrootd")
  The protocol from `xrootd`, the e**X**tended **ROOT D**aemon, which is used within the ROOT and PROOF frameworks and in some other places.
- SRM
  Storage **R**esource **M**anager is a meta-protocol targeted at storage management systems, including many specific features like space tokens, lifetime and pinning.

  For the actual data transport another transfer-protocol (like gsiFTP or gsiDCAP) is used.

Having multiple SRM-doors per cluster or more than one NFS 4.1 (pNFS)-door per node is possible, but a somewhat advanced setup. All other door-types can easily occur multiple times per cluster and node.

File Hierarchy Provider

In order to make access to the actual data possible trough a usual file hierarchy and to allow files being identified via a pathname, dCache makes use of a special service, namely the file hierarchy provider.

In the past, the separate "`pnfsd`" was used for this but it has been replaced by "Chimera", which provides much better performance and more advanced features.

Chimera keeps track of all pathname↔PNFS-ID-mappings, the pool-locations of each file's replicas as well as all the other file-meta-data (for example file-times, traditional POSIX file permission modes, ACLs and PNFS-tags).

All this information is stored within a database, managed by one of the supported database management systems.

dCache's components can access the file hierarchy and its meta-data via the previously mentioned PnfsManager.

It is furthermore possible to access both via NFS-mounts (for historical reasons often at "`/pnfs/`").

PNFS-IDs

dCache itself uses in most places the so called PNFS-IDs to identify and manage the actual files.

Basically there are two types of PNFS-IDs:

- Legacy PNFS-IDs
  These are 96 bit numbers that were used with dCache's legacy file hierarchy provider `pnfsd`. They are still used in clusters that were migrated from `pnfsd` to Chimera.
- Current PNFS-IDs
  With Chimera a new number format was introduced, now having 144 bits.
  The name "PNFS-ID" was retained for historical reasons.

PNFS-IDs are usually written as hexadecimal numbers, for example:

`000100000000000152203B20` (legacy PNFS-ID)

`0000AF1AF0735E30448C87FAF373325DEDA0` (current PNFS-ID)

It is worth to note, that all replicas of a given file share the same PNFS-ID.

Databases

Apart from the actual data (which is stored as files in a filesystem), dCache uses a database in most cases to store its internal information (working- and state-data).

The following databases (assuming default configuration) may be typically encountered:

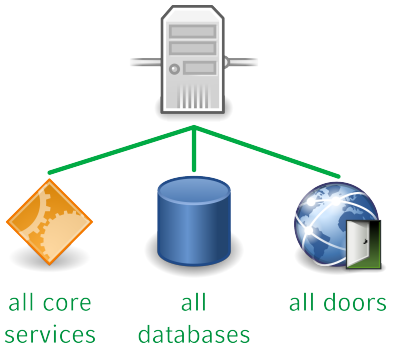
- `dcache`
  Holds miscellaneous data, including those from the PinManager and the SpaceManager as well as any `gplazma`-service(s) and SRM-door(s).
- `chimera`
  Holds the data from dCache's file hierarchy provider Chimera.
- `replicas`
  Holds the data from dCache's ReplicaManager.
- `billing`
  Holds the data from dCache's billing system.
  This is only used when `billingToDb` is set to `yes` in the configuration.

Currently, PostgreSQL is the only fully supported DBMS, but in the future generic database interfaces will be implemented that allow other DBMS being used.

Typical Cluster Setups – Core-Nodes

In a small cluster, the following setup might be used:
All core services, all databases and all used doors run on one single node.

+ Cheap as it requires only one node.
+ Might be easier to administrate.
- Puts a high load on this node.
- Definitely not suitable for “bigger” clusters and one could even consider whether a storage management system is the right choice.
- Single point of failure.
- Might be difficult to maintain, especially if more functionality or features shall be added.

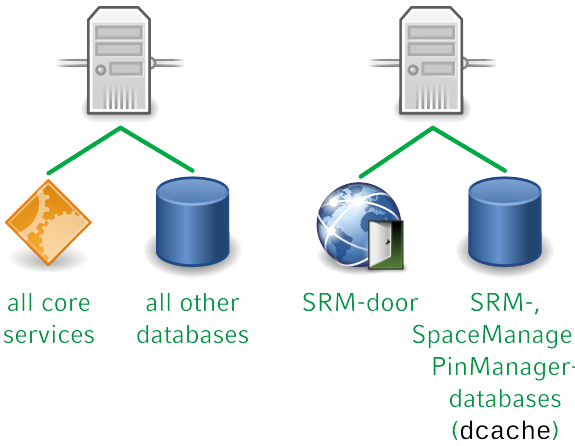


Typical Cluster Setups – Core-Nodes

In a medium cluster, the following setup might be used:
All core services and most databases run on one node, while the SRM-door and its related databases run on another.

+ "Reasonable" distribution of the load.
+ Great benefit for relatively few additionally required hardware-resources.
- There are still many components and probably large databases on the non-SRM-node.
- Only makes sense if SRM is heavily used.

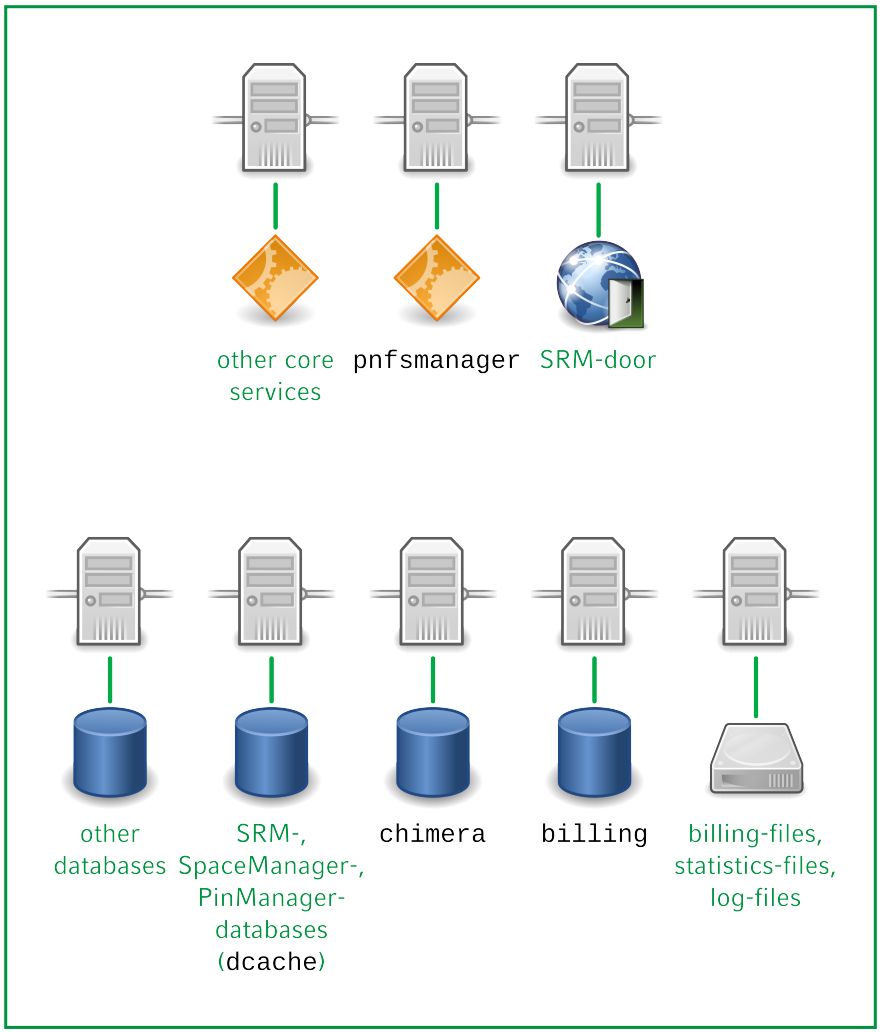
Analogous, the `pnfsmanager`-service and its related database (`chimera`) could be split off.



Typical Cluster Setups – Core-Nodes

In a large cluster, the following setup might be used:
Most core services run on one node, but the `pnfsmanager`-service has a dedicated node as well as the SRM-door.
The big databases have dedicated nodes, too.
The billing-files and statistics-files as well as the dCache-log-files from the miscellaneous nodes are written to a remote filesystem.

+ "Perfect" distribution of the load.
- Requires much hardware-resources "just" for the core-nodes.
- Might be difficult to administrate.
- Splitting off the SRM-door and its related databases only makes sense if SRM is heavily used.



Typical Cluster Setups – Door-Nodes

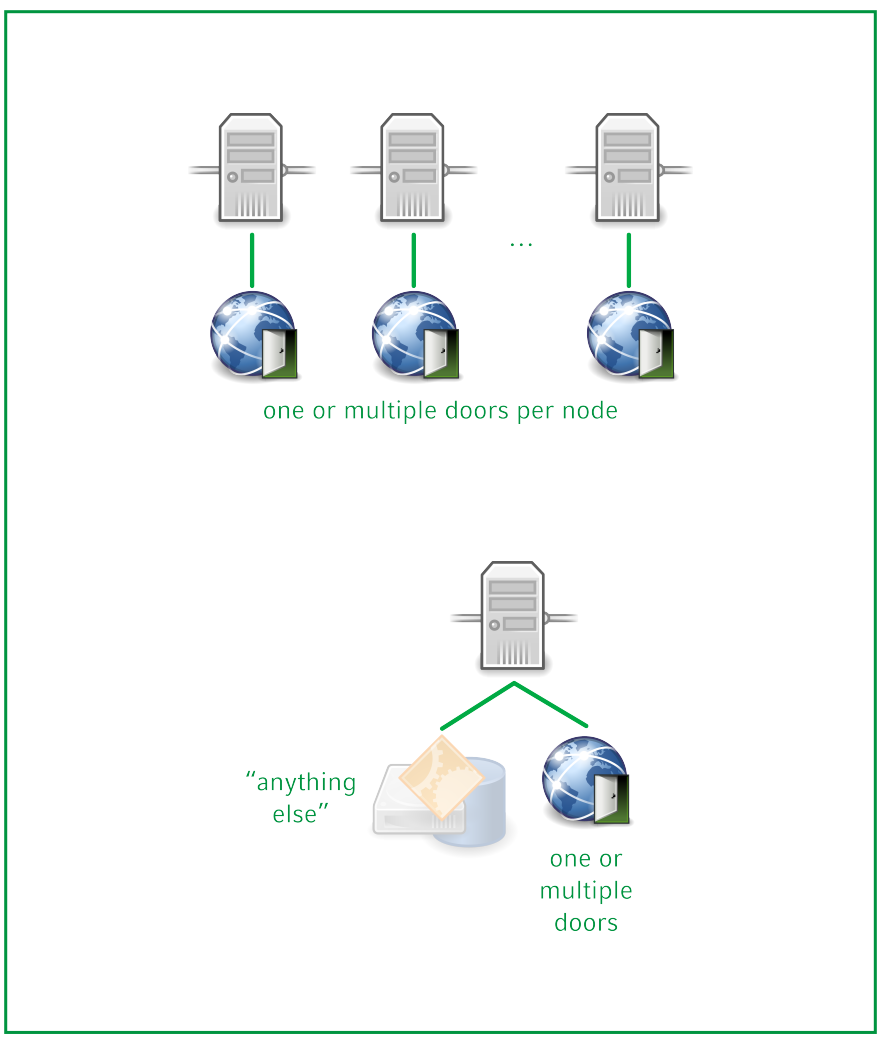
There are basically two ways to run a door:

- Using a dedicated node.
- Attaching it to "another" node, for example a core- or a pool-node.

In addition, it is possible in both cases to have either one or multiple doors per node.

The answer to these choices depends on at least the following factors:

- Is the specific protocol heavily used or not?
- Produces the door such a high load that it needs a dedicated node or can it coexist with the services from another node?
- Flows the actual data directly between the client and the pool or indirectly through the door (thus producing a high network load)?



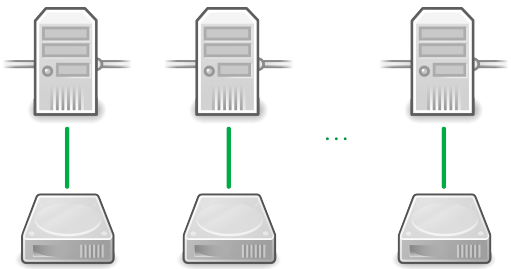
Typical Cluster Setups – Pool-Nodes

By their nature, pools run usually on dedicated nodes (the storage-nodes).

It is however possible to run one or multiple pools per node.

The answer on the “best” pool-allocation depends on at least the following factors:

- Usually, there should be one pool per volume (for example a single hard disk or the storage exported by a RAID-controller).
- Pools should however be not to small and not to big, which is why it can make sense to unify or to split into logical volumes, each running separate pools.
- Administration of the cluster might be easier, if all pools have similar sizes.

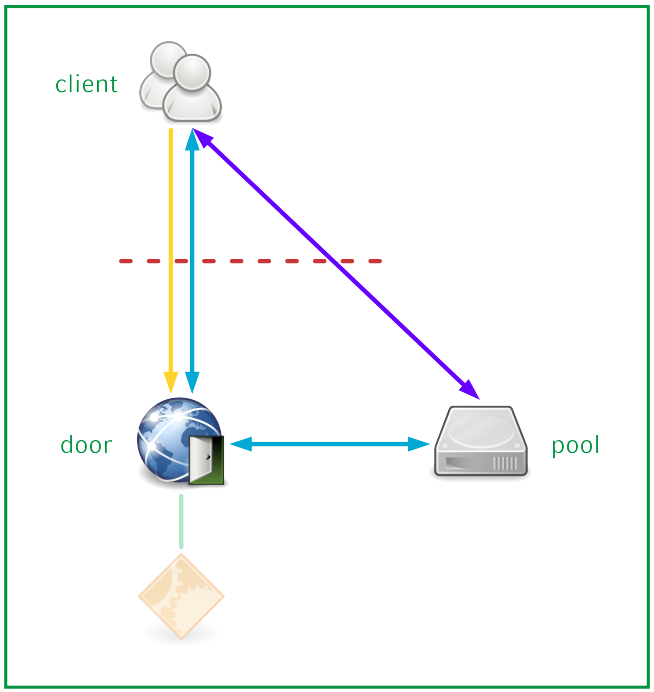


one or multiple pools per node

Data Access Model

The following is a very brief description of the access-process, the procedure that happens when a client reads or writes data to the cluster:

1. The client connects to a door with the desired protocol.
2. dCache determines the pool from which data should be read or where the data should be written to.
3. Depending on the protocol, its configuration and the client's request the actual data flows either directly between the client and the pool or indirectly via the door.
   As far as new connections are opened, these may be initiated either by the client or the server (which can be the door or the pool).



Of course there are many other steps like the authentication- and authorisation-process, accounting, et cetera.

Data Access Model

Further inherent characteristics of dCache's data access model:

- no random write-access
  Files can be written only sequentially (or overwritten completely) and not randomly as it is usual with "normal" filesystems.
  While this sounds like a big limitation, it is perfectly enough for many application scenarios in which files are written once and read many times.
- no striping of files
  At least until now, files are not striped over multiple pools.