Using Public dCache

Last Revised: November 29, 2015

Table of Contents

Chapter 1: Introduction to dCache

dCache is a distributed, multi-petabyte scalable disk storage system with a single rooted filesystem providing location independent file access. dCache can be used stand-alone or configured as disk cache front-end to a tertiary hierarchical storage management (HSM) systems (e.g., hard disk and tape) for I/O optimization. dCache is a joint venture between the Deutsches Elektronen-Synchrotron, DESY, the Fermi National Accelerator Laboratory, FNAL and the Nordic Data Grid Facility, NDGF.

dCache separates the namespace of its data repository from the actual physical location of the files. The namespace is internally managed by a database and is interfaced to the user application via the NFS protocol and via various FTP and SRM namespace operations. The location of a particular file may be on one or more dCache data servers (a.k.a data pools) as well as within the repository of external tertiary storage manager. dCache transparently handles all necessary data transfers between the pools and optionally between the external HSM and the pools. Internal pool-to-pool, or HSM-to-pool transfers may be caused by configuration or load-balancing constraints. When a file is transient, all dCache client operations to the file are suspended and resumed as soon as the file is fully available.

As the result of namespace and data separation, dCache pools can be added at any time without interfering with system operation. Having an HSM attached or having the system configured to hold multiple copies of each file, means that the pool nodes can be shut down at any time. In both setups, dCache is tolerant against failures of its data pool nodes.

Files written to dCache are *immutable*, meaning that once written files cannot be modified. The minimum data unit handled by dCache is a file. File resilience is achieved by optionally replicating the whole file across multiple pool nodes or by keeping a copy in tertiary storage. No file striping is supported.

dCache supports the following file access protocols:

- dCache native access protocol, dcap, supporting regular file access functionality. The dcap software package includes c-language client implementation of this protocol offering POSIX *open, read, write, seek, stat, close* operations as well as standard filesystem namespace operations. The provided library may be linked against client applications or may be preloaded to intercept file system I/O calls. The library supports security plugins. Currently GSS(Kerberos), GSI and SSL security plugins are provided. Additionally it performs all necessary actions to survive network or pool node failures. It allows the user to open files using an URL like syntax eliminating the requirement to have the dCache namespace mounted via NFS on the client host.
- FTP. Password authenticated or GSS(Kerberos) authenticated.
- GSI FTP (a.k.a. GridFTP) protocol version V1 and V2. dCache has a native implementation of the GridFTP protocol.

- SRM (version 1 and 2)
- XRootD protocol
- HTTP(s) and WebDAV
- NFS
    - v2 and v3 w/o file I/O. POSIX I/O is possible with the dcap preload library.
    - v4.1 (pNFS). NFS with POSIX file I/O with parallel connections to data pools

I/O protocols are provided by I/O servers, called *doors* in dCache. dCache doors are protocol converters, i.e. they convert a protocol-specific sequence of commands into a sequence of internal-to-dCache messages between its components resulting in protocol-specific replies back to the clients connected to the doors. Whenever an application needs to access files in dCache, it has to choose an appropriate door into the system. Each experiment determines which door(s) to use, and communicates this information to the Enstore administrators who manage the doors' configurations. (Enstore provides distributed access to and management of data stored on tape.)

dCache can be connected to one or more HSMs. In order to interact with an HSM, an external procedure has to be provided to handle restore/store/remove data in the HSM. dCache provides standard methods to optimize HSM access. Whenever a file is requested that cannot be found on any of dCache pools, a request is sent to the connected HSM to retrieve the file. When a file is retrieved (restored), it is made available to the requesting clients. To select a pool for staging a file, the system considers a variety of factors such as configuration information as well as pool load, available space and an LRU (Least Recently Used) algorithm to vacate space on pools for incoming files. The files written into dCache by the client is collected and, depending on the configuration, is flushed to the HSM based on policies that allows grouping of the "same" data on a tape or a particular set of tapes. Space management is handled internally by dCache. Replicas of files that exist on permanent storage will be removed from pools based on LRU only when new space is needed by incoming files.

While dCache distributes files autonomously across its data pools, the data flow preferences can be configured based on a set of rules that may take into account data flow direction, the sub-directory location within the dCache filesystem, storage information of the connected HSM as well as client IP or transfer protocol.

The dCache load balancing module plays a role in the pool selection process. It keeps itself updated about the number of active data transfers and the age of the least recently used file in each pool. Based on this information, the best pool to place data is chosen. The system is efficient even if requests arrive in bunches. Pools may be configured to initiate pool-to-pool transfer to less loaded pools to smooth out the overall load. Pools even can re-fetch the data from HSM rather than from other pools if all pools holding the data are too busy. Safeguards are in place to prevent chaotic pool-to-pool transfers when the global load is steadily increasing. Furthermore, the maximum number of file replicas on pools can be limited to avoid having the same set of files on all pools.

dCache can be configured to operate the Replica Manager for file resilience against pool node failures. Replica Manager enforces that at least N, but not more than M, copies of each file in a set of pools is maintained.

More general information about the dCache is available at http://www.dcache.org/.

Currently, dCache is the solution of choice for handling high data volumes produced by HEP experiments. Fermilab operates several dCache instances attached to Enstore HSM.

- CDF dCache (cdfen)
- CMS dCache
- D0 dCache.
- Public dCache (fndca)

Public dCache is used by Intensity Frontier (IF), LQCD and astrophysics groups for data storage. In the Fall of 2013 the system was expanded to meet the increasing demand for storage in terms of capacity and I/O throughput. In combination with Enstore, the system provides seamless access to tens of Petabytes of data stored on tapes.

Chapter 2: Chimera Namespace

The dCache namespace is shared by both dCache and Enstore. The implementation of the namespace is called Chimera. It presents files stored in the system in a directory tree structure. The namespace can be exposed to clients via a NFS mount. dCache supports NFS v2, v3 and v4.1 (pNFS) versions. NFS v4.1, which provides POSIX file I/O access,can be mounted only on the hosts running SLF6 or later. NFS v2 and v3 variants allow only access to file metadata and filesystem directory functions (unless the dcap preload library is used).

In addition to regular file metadata, the namespace stores storage-specific and other metadata in special files, called directory tags and file layers.

To browse file entries in the dCache system, on-site users can mount their experiment's namespace storage area on their own computers, and interact with it using standard UNIX operating system utilities. Normal UNIX permissions and administered export points are used to prevent unauthorized access to the name space.

Additionally the namespace can be browsed using the following clients:

- WebDAV
- FTP
- SRM
- xrd - xrootd file and directory meta-data utility

2.1 UNIX commands that can be used on Chimera namespace

As it has been noted above, the dCache namespace can be exposed to the client via a NFS mount. NFS v2, and v3 mounts do not allow data I/O. Therefore the commands such as c*at, more, less, head, grep, head, tail, wc, od, file, cp* would not work. They fail with an I/O Error.

However, virtually any non-I/O UNIX command can be used in the /pnfs namespace. For help with these commands, consult a UNIX manual or the man pages. Read and write access is

governed by standard UNIX file permissions.

The file I/O commands with NFS v2 and v3 are enabled by utilizing the dcap preload library by setting:

```
% export LD_PRELOAD=/usr/lib64/libpdcap.so
```

or, if SLF6 node is available, by mounting the namespace as NFS v4.1, e.g.:

```
% mount -t nfs4 -o minorversion=1 \

pnfs-stken.fnal.gov:/pnfs/fs/usr/minos /pnfs/minos
```

The latter provides native POSIX I/O with data transfer between the client and pool nodes directly. Commands like c*at, more, less, head, grep, head, tail, wc, od, file, cp* will work fine. Since files in dCache are immutable, modifying file content (by editing or appending) is not allowed. If a file needs to be modified, the best strategy is to copy it to local disk, modify it, remove it from the dCache namespace and then copy the modified local file back to dCache.

 Note that the *mv* command merely renames the file path, no actual data is being moved. Therefore one cannot “move” a file from the scratch pool area to the tape-backed area and expect to have this file written to tape. Use *cp* instead.

The paths: **/pnfs/xyz**, **/pnfs/fs/usr/xyz** and **/pnfs/fnal.gov/usr/xyz** all refer to the same directory. When using Enstore without dCache the first path is most often used. When using dCache (with or without Enstore) the second path is most often used. The third path type is used with SRM transfers.

2.1.1 Limitations

There is no hard limit on the number of files in any given namespace directory, but it is recommended to keep the number of files under 2000. This is recommended for any NFS-based file system.

The maximum name length of an entry in the namespace is 256 characters. However, **encp** will further restrict the filename length to 200 characters.

2.2 About directory tags

dCache steering and Enstore specific configuration information is contained in special tag files (historically named 'pnfs tags') . In the Chimera namespace, each directory can have a number of tags. These directory tags may be used within dCache to control file placement policies in the pools and are used by Enstore for similar purposes (e.g. to determine tape library, tape set and number of tape drives to be used in parallel when storing files to tape).

When a new directory in the /pnfs namespace is created, it inherits references to the tags of its parent directory.

 The values of the tags in a given directory will be inherited only by **newly** created sub-directories in this directory. Tags in existing sub-directories will not be affected. Manually setting a directory’s tags will destroy references to its parent directory’s tags. This may be what you want to do, but be aware. Likewise, only newly written files into this directory or in newly created sub-directories of it will be affected by changed tags.

2.2.1 Supported tags

The supported tags include:

- storage_group - *String* - This tag determines the storage group associated with all files in this directory, and shows up as your experiment's top level directory under `/pnfs`. Each experiment or research project is assigned a unique storage identifier by the Enstore administrators. Enstore uses the storage group names to control and balance assignment of resources, such as tape drives and media, among the experiments. Each storage group is assigned an area in namespace, e.g., an experiment XYZ might be assigned the storage area `/pnfs/xyz`. Typically, one storage group is associated with an entire experiment. A storage group is assigned to each experiment by the Enstore administrators. Users never change this tag.
- file_family – *String* - This tag determines the file family associated with all files in this directory. This is an Enstore specific flag. Files are grouped on Enstore tape volumes according to the storage group and + file family attribute. A file family is a name that defines a category, or family, of data files. Each experiment (i.e., each storage group) must carefully plan its set of file families. There may be many file families configured; by design there is no pre-set upper limit on the number. A given storage volume may only contain files belonging to one file family.
- file_family_width - *Integer* - This tag determines the file family width associated with all files in this directory. File family width is an integer value associated with a file family that is used to limit write-accessibility on data storage volumes; there is no width associated with reading. For a given media type and for a given file family, Enstore limits the number of volumes available for writing in parallel to the value of the file family width (except when unfilled volumes are already mounted for previous reads). Correspondingly, the number of media drives on which the volumes are loaded is also limited to the width.
- file_family_wrapper – *String* - This tag determines the file family wrapper associated with all files in this directory. A file family wrapper specifies the format of files on the storage volume. It defines information that gets added before and after data files as they're written to media. In this way the data written to tape is self-contained and independent of metadata stored externally. There are two wrapper types implemented, `cpio_odc, and cern`. The `cpio_odc` wrapper is the default wrapper set up by the Enstore admin when a new namespace area is created. All files with the `cpio_odc` wrapper can be dumped with **`cpio`**. This wrapper has a file length limit of (8G – 1) bytes. It is sufficient for the vast majority of data files, as most files are still under 2GB. The `cern` wrapper accommodates data files up to (10^21 –1) bytes, which in effect limits the file size to the tape size, since spanning and striping of files across multiple tapes are not supported. It matches an extension to the ANSI standard, as proposed by CERN, and allows data files written at Fermilab to be readable by CERN, and vice-

versa.

- Library - *String* - This tag determines name of the tape library associated with the data in the directory. Library is the name of Enstore logical tape library server. Users never change this tag.
- CacheClass – *String* - This is optional dCache specific tag that allows an additional selection dimension to direct data flows (in addition to storage group and file family).
- RetentionPolicy – *String* - This is optional dCache specific tag. It determines quality of data retention or in other words whether or not the data in the directory containing this flag will go to tape or not. Possible values are CUSTODIAL or REPLICA. Files written to directories having this tag set to CUSTODIAL will be written to tape else, they will stay on pools only.
- AccessLatency – *String* - This is optional dCache specific tag. It determines availability of the data files. Possible values are ONLINE and NEARLINE. Replicas of files written to directories having this tag set to ONLINE will always be available in dCache disk pools wheres files written to directories with this tag set to NEARLINE will be subject to LRU based sweeping once they were written to tape.
- WriteToken - *String* - This is optional dCache specific tag that determines in what space token to write the data in this directory. Only used if SRM space management feature is enabled.

Directory tags are an optional feature for dCache and are relevant only for Enstore. dCache can be setup to use a combination of storage_group, file_family and cacheClass to setup data steering policies.

2.2.2 Create list and read directory tags on mounted namespace

If the Chimera namespace is mounted, change to the directory for which the tags should be set and create/modify a tag with the following commands

```
% cd <directory>

% echo '<content1>' > '.(tag)(<tagName1>)'

% echo '<content2>' > '.(tag)(<tagName2>)'
```

Then the existing tags can be listed like this:

```
% cat '.(tags)()'
```

And the content of the tag can be read with:

```
% echo '<content1>' > '.(tag)(<tagName1>)'
```

A nifty way to list all tags with their content:

```
% grep “” $(cat “.(tags)()”)
```

Example:

```
% cd /pnfs/fs/usr/minos

% grep "" $(cat ".(tags)()")
        .(tag)(file_family):minos
        .(tag)(file_family_width):1
        .(tag)(file_family_wrapper):cpio_odc
        .(tag)(library):CD-LTO4F1
        .(tag)(OSMTemplate):StoreName sql
        .(tag)(sGroup):chimera
        .(tag)(storage_group):minos
```

 When creating or changing directory tags one has to keep in mind that the tags are not regular files, because the tags are different in the following aspects:

1. <*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.
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. Creating a new primary tag in a directory will not create an inherited tag in its subdirectories.
4. Moving a directory within the Chimera namespace 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.
5. Empty tags are ignored.

2.3 Special commands

2.3.1 File layers

In addition to tags there are special layer files associated with each file in dCache namespace. In total there are 7 layers, but only 3 are in use. The layers are special files that may contain additional metadata associated with a file. The content of layer files can be obtained by executing the following command:

```
% cat “.(use)(N)(foo)”
```

Where **N=1,2,4** and **foo** is the file name. When using full path the command may look like

```
% cat “/pnfs/bar/.(use)(N)(foo)”
```

or

```
% cat /pnfs/bar/”.(use)(N)(foo)”.
```

When a file is written to the system, its layer 2 is filled with some internally relevant information. The layer 2 is filled for all files in dCache (except 0 length files written via NFS, which is a bug that will be fixed). The layers 1 and 4 are filled when file is written to tape. Therefore files that are not destined to tape, like files in scratch and precious dCache space, will not have layer 1 and layer 4 filled.
Layer 1 contains a single string, called a Bit File ID or BFID. Bit File ID as a unique identifier of thefile in Enstore tape system.
Example:

```
% cat ".(use)(1)(xi_sum.tgz)"

CDMS127714742100000
```

The bfid in above case is CDMS127714742100000. And example of accessing layer 4 :

```
% cat ".(use)(4)(xi_sum.tgz)"
VP1248
0000_000000000_0000191
1654912463
litvinse
/pnfs/fnal.gov/usr/test/litvinse/atom/disk01/g2/dmitri/xi_sum/xi_sum.tgz

000500000000000006036C8

CDMS127714742100000
stkenmvr147a:/dev/rmt/tps0d0n:1310154418
476631428
```

Line by line output shows :

1. enstore tape label,
2. location of file on the tape (location cookie),
3. file size in bytes
4. file_family
5. original file name
6. blank line
7. pnfsid
8. blank line
9. BFID
10. drive name
11. Adler32 CRC value in decimal.

 A presence of layers 1 and 4 on a tape bound file indicates that the file has been written to tape successfully.

2.3.2 Dot commands

In addition to querying layers using ".(use)(N)" commands there are special command than allows to query information about the files or interact with the system:

- Querying internal id, the pnfsid:

```
% cat ".(id)(F10000170_0002.mdaq.root)"

0000DB02E2F4B1714F959F82394665C1DF44
```

- Querying file checksum:

```
% cat ".(get)(F10000170_0002.mdaq.root)(checksum)"

ADLER32:0e5e230e
```

- Querying file locality

```
% cat ".(get)(F10000170_0002.mdaq.root)(locality)"

ONLINE_AND_NEARLINE
```

File locality expresses file accessibility. The following values and their meaning are summarized in Table 1

File locality	Disk	Tape
ONLINE	Available in dCache disk pool	Not on tape
NEARLINE	Not available in dCache disk pool (needs staging for open)	Available on tape
ONLINE_AND_NEARLINE	Available in dCache disk pool	On tape
UNAVAILABLE	• File is in dCache disk pool, but pool is down • File is not on any of dCache disk pools	Not on tape

Table 1: Meaning of file locality in dCache

In addition to querying data, it is possible to pin or unpin files on dCache disk pool for a certain duration by utilizing the ".(fset)" command :

touch ".(fset)(*filename***)(pin|stage|bringonline)(***duration***)[(SECONDS|MINUTES|HOURS|DAYS)]"**

The last argument is optional and defaults to SECONDS. A duration value of 0 will unpin the file. Example:

```
% touch ".(fset)(F10000170_0002.mdaq.root)(pin)(7)(DAYS)"
```

or equivalent:

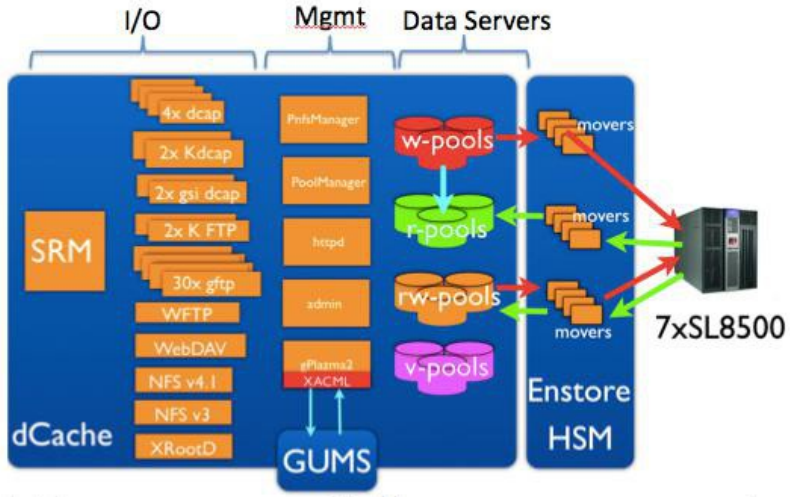
```
% touch ".(fset)(F10000170_0002.mdaq.root)(pin)(25200)"
```

will pin a file in dCache pool for one week. To unpin:

```
% touch ".(fset)(F10000170_0002.mdaq.root)(0)"
```

Chapter 3: How to use Public dCache

3.1 Introduction to Public dCache



Ilustration 1: Architecture of Public dCache/Enstore system

Architecture of general purpose Fermilab Public dCache system is depicted in the illustration. The system consists of three distinct layers:

- An I/O interface, which is collections of dCache *doors*, each of which implements a specific I/O protocol and appropriate authentication mechanism. While some doors are unique (like the SRM door), multiple doors of the same protocol can be set up providing load balancing and resilience against hardware failures as well as ability to perform rolling software upgrades.
- A management and monitoring layer consisting of the following services:
  - A pluggable grid-enabled authorization module (gPlazma) which calls out site-wide GUMS server to obtain DN → local user and ultimately local UID:GIDs mapping
  - A PoolManager, a service controlling data flows to/from pools or the Enstore HSM
  - A PnfsManager, a namespace provider service that is responsible for file ID to path resolution and permission handling.
  - An httpd service that runs on port 2288 and shows a live view of the system (at http://fndca.fnal.gov:2288/)
  - An Admin interface
- Data servers, known as data pools in dCache. A dCache pool is a service running on a data server that manages a directory containing file replicas. Each pool communicates with Enstore systems by means of an HSM interface which invokes system calls to a wrapper around the Enstore copy client "**encp**" when a

file store/restore or removal is requested.

☞ The main HTML dashboard of Public dCache system is located at http://fndca.fnal.gov/

3.2 Using dCache to Copy Files

Whenever a client application needs to talk to the dCache, it has to choose an appropriate *door* into the system. For each door, there are corresponding utilities for copying files back and forth between the host and the **/pnfs/<storage-group>** area in dCache. The list of dCache doors, protocols that they implement, and clients that can be used to access the data is presented in Table1.

Protocol	Host:port(s)	Authentication	Client(s)
dcap	dcap://fndca1.fnal.gov:{24125,24136,24137,24138}	None	dccp,dcap, root
	dcap://fndca1.fnal.gov:{24525,24536}	GSI	
	dcap://fndca1.fnal.gov:{24725,24736}	Kerberos(GSS)	
FTP	fndca1.fnal.gov:24126	passwd	ftp
	fndca1.fnal.gov:24127	kerberos(GSS)	
GridFTP	fndca1.fnal.gov:{2811,2812}	GSI	globus-url-copy, srmcp, uberftp
SRM	srm://fndca1.fnal.gov:8443	GSI	lcg-cp, srmcp…
NFS v4.1	stkensrv1n.fnal.gov:2049	None	POSIX I/O
NFS v3	stkensrv1n.fnal.gov:2049	None	POSIX metadata ops, POSIX I/O with dcap preload library
WebDAV	https://fndca1.fnal.gov:2880	GSI	browsers, curl, cadaver, davfs, root
XRootD	(x)root://fndca1.fnal.gov:1094	GSI	xrdcp, xrd, POSIX preload library, FUSE, root

Table 2: List of dCache protocol specific I/O doors and supported clients

3.2.1 dCache access protocol, dcap

The dCache native access protocol, dcap, supports regular file access functionality. The dcap software package includes a c-language client implementation of this protocol offering POSIX *open*, *read*, *write*, *seek*, *stat*, *close* operations as well as standard filesystem namespace operations. The provided library may be linked against a client application or may be preloaded to intercept file system I/O calls. The library supports security plugins. Currently GSS (Kerberos) and GSI security plugins are provided. Additionally it performs all necessary actions to survive network or pool node failures. It allows the user to open files using an URL like syntax eliminating the requirement to

have dCache namespace mounted via NFS on the client host.
The dcap client, **dccp,** and API library are distributed as a standard package available from EPEL (Extra Packages for Enterprise Linux) yum repository. Installation involves the following sequence of steps.
For SLF5 as root:

```
% rpm -i \
  http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
% yum install --enablerepo=epel* dcap*
```

For SLF6 as root:

```
% rpm -i \
  http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
% yum install --enablerepo=epel* dcap*
```

Additionally, the dcap product is available in KITS at ftp://fnkits.fnal.gov/products/dcap. Installation usually involves the following steps:
As user products

```
% . /fnal/ups/etc/setup.sh
% setup upd

% upd install dcap -G"-c"
```

Then, as regular user :

```
% . /fnal/ups/etc/setup.sh

% setup dcap
```

Use

```
% man dccp
```

or

```
% dccp –-help
```

to get help on dccp usage.

3.2.1.1 Authentication Mechanisms

There are three authentication mechanisms used for the dcap protocol: "plain", kerberos, and X509. When using dcap from KITS, all three have separate "setup dcap" qualifiers for the UPS/UPD distribution of dcap.
These different qualifiers have to be setup correctly in UPS for this to work though, with a ups listing for each qualifier state. Different setups define environment variable DCACHE_IO_TUNNEL to point to different shared libraries implementing authentication mechanism: libgsiTunnel.so for x509, libgssTunnel.so for kerberos and unset for "plain" (unauthenticated) dcap access.

Plain dcap – no authentication

Plain dcap is strictly limited to fnal.gov domain access only. It uses uid/gid permissions on files in Chimera namespace. Plain dcap doors run on ports 24125,24136,24137,24138 on fndca1.fnal.gov. The UPS setup command reads:

```
% setup dcap -q unsecured
```

Alternatively, if dcap package is deployed on your system as RPM, nothing needs to be setup.

Kerberos authenticated dcap (or kerberized dcap)

Kerberized dcap doors run on ports 24725,24736 on fndca1.fnal.gov. To be able to connect to kerberized dcap door a client has to obtain kerberos ticket from FNAL KDC like so:

```
% kinit <username>@FNAL.GOV
```

Invoke UPS setup command:

```
% setup dcap
```

Alternatively, if dcap package is installed on your system as RPM, then the environment variable needs to be defined:

```
% export
DCACHE_IO_TUNNEL=/usr/lib64/dcap/libgssTunnel.so
```

before a connection to a kerberized dcap door can be made.

X509 authenticated dcap or GSI dcap

GSI dcap doors run on ports 24525,24536 on fndca1.fnal.gov. To be able to connect to GSI dcap door a client has to obtain kerberos ticket from FNAL KDC and convert it into X509 compliant proxy following the following sequence of commands.

```
% kinit <username>@FNAL.GOV

% kx509
```

Alternatively **grid-proxy-init** or **voms-proxy-init** can be used to obtain grid or voms proxies:

```
% grid-proxy-init
```

```
% voms-proxy-init
```

Invoke UPS setup command:

```
% setup dcap -q x509
```

Alternatively, if dcap package is installed on your system as RPM, then the environment variable needs to be defined:

```
% export
DCACHE_IO_TUNNEL=/usr/lib64/dcap/libgsiTunnel.so
```

before a connection to a GSI dcap door can be made.

3.2.1.2 The dccp command

The command **dccp**, which provides a **cp**-like functionality on the Chimera file system, is available as part of dcap product (or RPM package). The **dccp** command has the following syntax:

```
% dccp [options] <src> <destination>
```

Where source is path to a local file and destination is path to the destination in Chimera namespace. Destination can either be an URI or a path in Chimera namespace if it is mounted on the local host. URI has the form (all on one line):

```
dcap://<serverHost>:<port>/</pnfs>\
/fnal.gov/usr/<storage_group>/<filePath>
```

Where serverHost is fndca1.fnal.gov, port is one of the ports specified in Table1 for dcap protocol.
In addition to the **dccp** command, the Fermilab dcap product available from KITS provides two other useful scripts **dc_stage** and **dc_check** described below.

3.2.1.3 The dc_stage Command

The **dc_stage** command pre-stages a file for read requests only. It is particularly useful when you'd like to grab the file quickly from the dCache when you're ready for it. Use this with the **-t** option to set an interval of time between the download to the dCache and the download from the dCache to your local system. If **-t** is not used, the default interval is zero.

```
% dc_stage <filename>
```

The **dc_stage** script is not supplied with dcap RPM, instead use the following equivalent invocation of **dccp**:

```
% dccp -P <filename>
```

The above commands are non-blocking meaning that the client makes a connection to the dCache server and requests a file pre-stage. It returns as soon as the system has successfully scheduled the pre-stage request but not when the file has actually been staged in.

3.2.1.4 The dc_check Command

The **dc_check** command checks if a file is on disk (or "on-line") in the dCache.

```
% dc_stage <filename>
```

The **dc_check** script is not supplied with dcap RPM, instead use the following equivalent invocation of **dccp**:

```
% dccp -P -t -1 <filename>
```

Check the return code. 0 means the file is on-line, else the file is off-line (needs pre-staging).

3.2.1.5 Syntax and Examples (PNFS Not Mounted Locally)

If the Chimera namespace is not mounted locally (the general case), you'll have to supply the protocol, node, port, and pnfs directory for the remote location (the "source" on reads, and the "destination" on writes). For example, a command requesting a write to dCache would have this structure:

```
% dccp path/to/local/file \

dcap://fndca1.fnal.gov:24125//pnfs/fnal.gov/usr/\

<storage_group>/<filePath>
```

Run **dc_check** to check that the file is on-line now:

```
% dc_check \

dcap://fndca1.fnal.gov:24125//pnfs/fnal.gov/usr/\

<storage_group>/<filePath>
```

or equivalently:

```
% dccp -P -t -1 \

dcap://fndca1.fnal.gov:24125//pnfs/fnal.gov/usr/\

<storage_group>/<filePath>
```

The command to read a file would look like:

```
% dccp \
```

```
dcap://fndca1.fnal.gov:24125//pnfs/fnal.gov/usr/\
<storage_group>/<filePath> \
 path/to/local/file
```

To pre-stage the file:

```
% dc_stage \
dcap://fndca1.fnal.gov:24125//pnfs/fnal.gov/usr/\
<storage_group>/<filePath>
```

or equivalently:

```
% dccp -P -t -1 \
dcap://fndca1.fnal.gov:24125//pnfs/fnal.gov/usr/\
<storage_group>/<filePath>
```

The above examples use plain dcap. The command syntax is identical when using strong authentication. The difference is – door ports and the requirement to set the DCACHE_IO_TUNNEL environment variable. Refer to **Table1** for GSS and GSI port values and look up how to set up strong authentication in *Authentication Mechanisms* section.

3.2.1.6 Syntax and Examples (PNFS Mounted Locally)

If the Chimera namespace is mounted on your local machine via NFS v2 or v3, you only need to specify the simple Chimera path of the remote file, e.g. (for a write):

```
% dccp path/to/local/file \
/pnfs/fnal.gov/usr/<storage_group>/<filePath>
```

The command to read a file would look like:

```
% dccp \
 /pnfs/fnal.gov/usr/<storage_group>/<filePath> \
 path/to/local/file
```

Note that the /pnfs file path depends on the name of the mountpoint as mounted on your system. Usually it is **/pnfs/**<**storage_group**>/<**filePath**>

3.2.1.7 Syntax and Examples: dcap pre-load library

If the Chimera namespace is mounted on your local machine via NFS v2 or v3, it is possible to use Unix POSIX I/O calls with the *dcap preload library.* Define this variable

to point to the dcap preload library on your system:

```
% setup dcap -q unsecured

% export LD_PRELOAD=$DCAP_DIR/lib /libpdcap.so
```

Now system I/O calls will be intercepted by dcap library calls and can use files in Chimera namespace as if they were normal files on a local partition. Due to the immutability of files in dCache, modification of files is still not allowed.
If your system has the dcap RPM installed, you can query for the location of the dcap library and define LD_PRELOAD environmental variable accordingly. E.g.

```
% rpm -qa | grep dcap-libs

dcap-libs-2.47.8-1.el5.x86_64

% rpm -ql dcap-libs-2.47.8-1.el5.x86_64
    /usr/lib64/dcap
    /usr/lib64/libdcap.so.1
    /usr/lib64/libdcap.so.1.1.47
    /usr/lib64/libpdcap.so.1
    /usr/lib64/libpdcap.so.1.1.47
    /usr/share/doc/dcap-libs-2.47.8
    /usr/share/doc/dcap-libs-2.47.8/AUTHORS
    /usr/share/doc/dcap-libs-2.47.8/COPYING.LIB
    /usr/share/doc/dcap-libs-2.47.8/LICENSE

% export LD_PRELOAD=/usr/lib64/libpdcap.so.1
```

3.2.1.8 Syntax and Examples: pnfs ''protocol''

All files in Chimera namespace receive unique IDs, called PNFSIDs for legacy reasons. You can look at the pnfsdid on the mounted Chimera namespace:

```
% pnfsid=`cat /pnfs/<storage_group>/path/to/".(id)
(file)"`
```

Then you can use the PNFSID to retrieve the file:

```
% dccp pnfs://fndca1.fnal.gov:24125/${pnfsid}\

path/to/local/file
```

3.2.2 Grid (GSI) FTP

GSI stands for Grid Security Interface. GSI FTP uses Grid Proxies for authentication and authorization and is compatible with popular Grid middleware tools such as **globus-url-copy** (from the Globus toolkit available at http://www.globus.org). The dCache GSI FTP currently runs on ports 2811 and 2812 on fndca1.fnal.gov (see Table1). Additionally all pool nodes in dCache run one GFTP door on default port 2811.
It is more convenient to run this through an interface like srmcp (see section Storage Resource Management (SRM) *)* which allows you to perform multiple transfers in a single command. In addition, it optimizes the parameters of the transfer, and allows FTP to scale with user load (overcoming a passive GFTP protocol issue).

3.2.2.1 Obtain Grid Proxies

Globus tools require that a user be authenticated with a short-term authentication Grid proxy. This proxy is created from (long-term) X509 certificates issues by DigiCert. We recommend that you use the command grid-proxy-init to generate your proxy from your certificate. A proxy expires after a preset duration, and then a new one must be regenerated from the user's (long-term) X509 certificate. Example:

```
% grid-proxy-init
      Your identity: /DC=com/DC=DigiCert-Grid/
         O=Open Science Grid/OU=People/
         CN=Dmitry Litvintsev 1123
         Enter GRID pass phrase for this identity:
         Creating
      proxy ........................................
      Done
         Your proxy is valid until: Tue Jul 15 09:13:29
      2014
```

X509 Grid proxies can be issued automatically for Fermilab users authenticated to Kerberos. This involves downloading a KX509 certificate. KX509 can be used in place of permanent, long-term certificates. It works by creating X.509 credentials (certificate and private key) using your existing Kerberos ticket. These credentials are then used to generate the Globus proxy certificate.

```
% kinit litvinse@FNAL.GOV

% kx509
      Service kx509/certificate
      issuer= /DC=gov/DC=fnal/O=Fermilab/OU=Certificate
      Authorities/CN=Kerberized CA HSM
      subject= /DC=gov/DC=fnal/O=Fermilab/OU=People/CN=Dmitry O.
      Litvintsev/CN=UID:litvinse
      serial=02C9F9EB
      hash=11248d6a
      Your identity: /DC=com/DC=DigiCert-Grid/O=Open Science
      Grid/OU=People/CN=Dmitry Litvintsev 1123
      Enter GRID pass phrase for this identity:
      Creating proxy ..................................... Done
      Your proxy is valid until: Tue Jul 15 09:13:29 2014
```

3.2.2.2 GSI FTP with globus-url-copy

Install the Globus toolkit (available from a variety of locations, http://www.globus.org is one). Then run the **globus-url-copy** command in order to use the GSI FTP protocol to transfer files. Use the **gsiftp://** URL prefix for the file path path, and **file://** for the other URL.
E.g., for copying a file from dCache to the local node

```
% globus-url-copy gsiftp:\

//[[<src_node>:]port]/<source_url_path> \

 file:///path/to/the/local/file
```

and to copy file to dCashe it is

```
% globus-url-copy file:////path/to/local/file
gsiftp://[[<src_node>:]port]/<destination_url_path>
```

Example:

```
% globus-url-copy gsiftp://fndca1.fnal.gov/<filePath>
file:///tmp/my_file

% globus-url-copy file:///tmp/my_file
gsiftp://fndca1.fnal.gov/my_file
```

The port was omitted implying default GridFTP port 2811. Refer to **globus-url-copy** documenation at this link
http://toolkit.globus.org/toolkit/docs/5.2/5.2.5/appendices/commands/#globus-url-copy

 For 3-rd party transfers, that is when both source and destination are URI of GridFTP servers like so :

```
% globus-url-copy gsiftp://fndca1.fnal.gov/my_file \
gsiftp://hostname:2811/foo
```

it is required to specify -**dodcau** option as dCache GridFTP implementation does not support data channel authentication.

3.2.3 Storage Resource Management (SRM)

Storage Resource Manager (SRMs) is a grid middleware component 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.The SRM standard allows independent institutions to implement their own SRMs. SRMs leave the policy decision to be made independently

by each implementation at each site. Resource Reservations made through SRMs have limited lifetimes and allow for automatic collection of unused resources thus preventing clogging of storage systems with “forgotten” files.

The storage systems can be classified on basis of their longevity and persistence of the data they store. Data can be considered to be temporary and permanent. For example disk caches might allow for spontaneous deletion of the files, while deletion of the file stored in a robotic tape storage can be very problematic. To support these notions, SRM defines three types of files and spaces: Volatile, Durable and Permanent. Volatile files can be removed by the system to make space for new files upon the expiration of its lifetime. Permanent files are expected to exist in the storage system for the lifetime of the storage system, unless explicitly deleted by the user. Finally Durable files have both a lifetime associated with them and a mechanism of notification of owners and administrators of lifetime expiration but can not be deleted automatically by the system and require explicit removal.

The SRM interface consists of the five categories of functions:

1. space management
2. data transfer
3. request status
4. directory
5. file/directory permission management

The SRM interface utilizes Grid Security Infrastructure (GSI) for authentications.

The SRM service is a Web Service implementation of a published WSDL document. SRM v2.2 specification can be found here https://sdm.lbl.gov/srm-wg/. The WSDL file is https://sdm.lbl.gov/srm-wg/srm.v2.2.wsdl.

Fermilab SRM implements a SRM v2.2 interface to dCache. For more information refer to https://srm.fnal.gov.

Srmcp is the implementation of the SRM client that provides data movement functionality similar to **globus-url-copy** but is different is several important aspects:

1. Provides end-to-end CRC selectable (currently supported MD5, MD4 and adler32) checks
2. It connects to dCache SRM server on a Site URL (**SURL**). The SRM server performs translation of SURL to transfer URL (**TURL**) taking into account system load and scheduling request if necessary.

Example of a SURL is

```
srm://fndca1.fnal.gov/pnfs/fnal.gov/usr/\
<storage_group>/foo
```

Example of a TURL is :

```
gsifdtp://stkendca19a.fnal.gov:2811/foo
```

Note that TURL is prefixed by protocol name, "gsidftp" in this case. The host part of TURL is selected by SRM by querying the dCache system internally for least used GridFTP door thus achieving load balancing. The path part of the TURL seems like a sub-path of the path parts of the SURL. This is because each user of SRM is mapped to an internal dCache user (there could be many users mapped to the same group account usually associated with the storage group). Each internal dCache user account has

associated root path which is treated by SRM as a chroot facility. Usually group accounts have **/pnfs/fnal.gov/usr/<storage_group>** root paths. When using srmcp, the translation is completely transparent to users and is used internally by srmcp. In fact srmcp encapsulates several SRM function invocations:

1. srmPrepareToPut{Get}. Asynchronous request to generate the TURL on a provided SURL.
2. srmStatusOf{Put,Get}Request. Periodically queries status of srmPrepareToPut{Get} request.
3. Once the TURL is obtained a GridFTP transfer is started on that TURL. The GridFTP transfer started by srmcp will be retried on transient errors until completed successfully.
4. In case of writing to srm (srmPut) srmPutDone is called signaling to the system that the transfer completed successfully.

3.2.3.1 Preparing to Use srmcp

To use the java-based **srmcp**, you will need to install java on your system. You will also need to install either the globus toolkit or dccp, depending on which protocol you wish to use. In order to use GSI with **srmcp**, follow the instructions in the README.SECURITY file that comes with srmcp in Kits.

3.2.3.2 Command Syntax

```
% srmcp [options] source(s) destination
```

Default options will be read from a configuration file but can be overridden by command line options. The options are listed and defined in the srmcp README file in Kits. We do not list them here.

```
% srmcp --help
```

Provides help on using the command.

3.2.3.3 Usage examples

As in case of GSI FTP or GSI dcap before using SRM the user has to generate on grid proxy:

```
% grid-proxy-init
```

Or:

```
% kx509
```

Or:

```
% voms-proxy-init
```

To copy a file from local disk to SRM :

```
% srmcp file:////path/to/local/file \

srm://fndca1.fnal.gov/pnfs/fnal.gov/usr/<storage_group
>/foo
```

To copy a file from SRM to local disk :

```
% srmcp srm://fndca1.fnal.gov/pnfs/fnal.gov/usr/\

<storage_group>/foo \

file:////path/to/local/file
```

Additionally srmcp allows the user to copy between two SRMs or between GridFTP doors. E.g. to copy a file between CDF dCache and Public dCache one has to issue :

```
% srmcp srm://cdfdca1.fnal.gov/pnfs/fnal.gov/usr/\

cdfen/filesets/foo \

srm://fndca1.fnal.gov/pnfs/fnal.gov/usr/\

<storage_group>/foo
```

3.2.4 FTP

Public dCache runs a plain FTP server and a Kerberized FTP server. See Table1 for port numbers.

3.2.4.1 Plain (aka weak) FTP

Password-based authentication. Example:

```
% ftp fndca1.fnal.gov 24126

Connected to fndca1.fnal.gov (131.225.13.30).
220 Weak FTP door ready
Name (fndca1.fnal.gov:litvinse):
331 Password required for litvinse.
Password:XXXXXX
230 User litvinse logged in
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
ftp> get fstab.s
local: fstab.s remote: fstab.s
227 OK (131,225,13,30,81,3)
150 Opening BINARY data connection for
/pnfs/fnal.gov/usr/test/litvinse/fstab.s
226 Transfer complete.
1266 bytes received in 0.00664 secs (190.55
```

```
Kbytes/sec)
ftp> quit
221 Goodbye
```

3.2.4.2 GSS (Kerberos) FTP

First, obtain kerberized FTP client. On SLF5.x system the kerberized FTP client is part of the pre-installed **krb5-fermi-base package**. The binary is **/usr/krb5/bin/ftp**. On SLF6.x system, the kerberized FTP client is provided by **krb5-appl-clients** package (**yum install -y krb5-appl-clients)**.

Authentication is based on kerberos ticket. Example:

```
% kinit <username>@FNAL.GOV

% ftp fndca1.fnal.gov 24127

Connected to fndca4a.fnal.gov.
220 Kerberos FTP door ready
334 ADAT must follow
GSSAPI accepted as authentication type
GSSAPI authentication succeeded
Name (fndca1.fnal.gov:litvinse):
200 User litvinse logged in
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get fstab.s
local: fstab.s remote: fstab.s
227 OK (131,225,13,30,80,17)
150 Opening BINARY data connection for
/pnfs/fnal.gov/usr/test/litvinse/fstab.s
226 Transfer complete.
1266 bytes received in 7.5e+02 seconds (0.0017
Kbytes/s)
ftp> quit
221 Goodbye
```

3.2.5 XRootD

XRootD is named after extended rootd protocol which provides POSIX-like random access to arbitrary data organized in files of any type (so it is not just root files). XrootD provides fault tolerant, low latency, high bandwidth access to data.

dCache provides a fully functional XRootD server using a native implementation of the

xrootd protocol in Java. It acts as any other dCache door and runs on node fndca1.fnal.gov, port 1094.

dCache XRootD uses GSI authentication. Therefore a client has to obtain a grid proxy first:

```
% grid-proxy-init
```

Or:

```
% kx509
```

Or:

```
% voms-proxy-init
```

Then use **xrdcp** to copy in/out of dCache using URI syntax:

```
% xrdcp \

xroot://fndca1.fnal.gov/pnfs/fnal.gov/usr/\

<storage_group>/foo path/to/local/file
```

Or use **xrd** file and directory manipulation utility.

```
% xrd fndca1.fnal.gov

(C) 2004-2010 by the Xrootd group. Xrootd
version: v3.0.2
Welcome to the xrootd command line interface.
Type 'help' for a list of available commands.
> cat /pnfs/fnal.gov/usr/<storage_group>/foo
```

3.2.6 Accessing files from root

Accessing files in dCache from a root application is very straightforward. One needs just to specify the protocol in the file URI. Examples:

Open root file via XRootD door:

```
% {grid,voms}-proxy-init (or kx509)

% root -l

root [0]
f=TFile::Open("root://fndca1.fnal.gov/pnfs/fnal.gov/usr/<st
orage_group>/foo")
```

Open root file via plain dcap door:

```
% setup dcap -q unsecured

% root -l
```

```
root [0]
f=TFile::Open("dcap://fndca1.fnal.gov:24125/pnfs/fnal.gov/usr/<storage_group>/foo")
```

Open root file via kerberized dcap door:

```
% setup dcap

% root -l

root [0]
f=TFile::Open("dcap://fndca1.fnal.gov:24725/pnfs/fnal.gov/usr/<storage_group>/foo")
```

Open root file via GSI dcap door:

```
% setup dcap -q kx509

% {grid,voms}-proxy-init (or kx509)

% root -l

root [0]
f=TFile::Open("dcap://fndca1.fnal.gov:24525/pnfs/fnal.gov/usr/<storage_group>/foo")
```

On SLF6 hosts, if the Chimera namespace is mounted as NFS v4.1, the URI can be replaced with just the full file pathnames of the NFS 4.1 mount on the client host.

3.2.7 WebDAV

Web Distributed Authoring and Versioning (WebDAV) is an extension of the Hypertext Transfer Protocol (HTTP) that allows users to create and modify web content. Many operating systems provide built-in client support for WebDAV. Information is available at: http://www.webdav.org/

dCache implements a WebDAV server as a dCache door running on port 2880. The dCache WebDAV door can be accessed by users having DigiCERT or KX509 certificates loaded in their browsers or using standard Linux clients like **wget**, or **curl**. WebDAV content can be mounted using the **davfs2** with fuse system modules, KDE has native WebDAV support. This enables Dolphin, Konqueror. and every other KDE application to interact directly with WebDAV servers. All applications using GIO, including Nautilus, have access to WebDAV through GVFS.

The WebDAV door is configured to use authenticated HTTP (HTTPS) protocol with GSI authentication module that requires users to provide X.509 certificate during login

procedure. Similarly to FTP, the WebDAV door uses the user root directory extracted from the storage-authzdb file to effectively chroot to it, thus exposing only files/directories belonging to this user.

3.2.8 Using browser with WebDAV door

First, the user certificate in PKCS12 format needs to be loaded into browser

```
% openssl pkcs12 -export -in \

./globus/usercert.pem -inkey \

.globus/userkey.pem -out cert.p12
```

For KX509 certificates:

```
% kx509 -o kca.crt

% openssl pkcs12 -export -in kca.crt -out cert.p12
```

The file cert.p12 needs to be loaded into a browser. After that type

https://fndca1.fnal.gov/2880

in the Browser's location bar. You should be able to see your directory tree. It is possible to download files. At the bottom you will find the “Browse” button which allows the user to upload files to existing directories.

Below is example of using **curl** with dCache WebDAV door:

```
% grid-proxy-init (or voms-proxy-init or kx509)

% curl -1 -L --capath \

/etc/grid-security/certificates \

  --cert /tmp/x509up_u8637 \


https://fndca4a.fnal.gov:2880/fermigrid/volatile/fermilab/litvinse/curl.txt\

   -o curl1.txt

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current

                                 Dload  Upload   Total   Spent    Left  Speed

100  1266  100  1266    0     0   4487      0 --:--:-- --:--:-- --:--:--  4487
```

3.3 Useful links

1. A FAQ in dCache is available at https://srm.fnal.gov/twiki/bin/view/DcacheCorner/DcacheFAQ
2. Fermilab dCache project home page is here: https://srm.fnal.gov
3. A global dCache home page is http://www.dcache.org