CDF/DOC/COMP_UPG/PUBLIC/5981
Version 2.01
20 September 2003

Introduction to DCache Use at CDF

Robert D. Kennedy
Fermilab Computing Division

Abstract

This note describes how one may use the dCache product from within the CDF Application Framework as a replacement for the CDF Disk Inventory Manager (DIM) and stager components of the CDF Data Handling system using the DHInputModules. This can be done on central CDF computing platforms (Linux and IRIX), CDF desktops, and the CDF CAF. This note is also a brief introduction to the dCache product, providing CDF-oriented examples of its different uses. This note version has been updated for cdfsoft2 release 4.11.2 and root v3.05/04, though most aspects are still applicable to earlier releases of cdfsoft2 and root.

Contents

List of Figures

1 Introduction to dCache

dCache is a product designed to be a front-end disk cache for large tape library systems [6, 8, 9, 10]. Originally designed and developed by DESY, dCache has been co-developed by DESY and the ISD department of FNAL for several years now.

This note is an introduction to dCache for CDF Offline software users. dCache can now be used in place of the CDF Disk Inventory Manager and stager software [2, 3, 4] with the either the DHInput modules. This note does not describe dCache use by SAM as that will be documented by the CDF-SAM team. Other CDF notes will describe the dCache system design as it matures. Much of the content here was originally based on an earlier CDF note [5].

Reaching the point where dCache can be used transparently by CDF Offline software was made possible by many contributors, including members of the FNAL CCF (formerly ISD) department (Jon Bakken, Rich Wellner, Don Petravick, Timur Perelmutov, and others), the dCache developers (Patrick Fuhrmann and his team at DESY), the CDF DH contributors (Rob Kennedy, Fedor Ratnikov, Jeff Tseng, Robert Harris, and Dmitri Litvintsev), and the authors of the TDCacheFile class (Grzegorz Mazur) and TPlugin classes (Fons Rademakers and the ROOT development team) in ROOT. The adaptation of the DHInputModules to use dCache in particular was accomplished by Fedor Ratnikov and Rob Kennedy.

1.1 DCache Design

dCache uses network-mounted disks to implement a distributed data cache with user authentication [7]. A dCache system consists of cache-wide services running on one or more administrative nodes, and one or more storage pools marked for read access and/or write access. The pools may all be on one or more hosts, even on admin nodes, but pools themselves do not span hosts. Each host supporting pools also runs pool-specific management services. Both volatile and static data files are supported in dCache pools. dCache systems depend on the presence of a mass storage system (tape library) from which to request files if they are not in cache.

Clients send requests for a datafile to a “door” of a dCache system. A door is a network server which performs user authentication and forwards client requests to the pool managers. There can be more than one door to a dCache system, each potentially handling a distinct authentication mechanism and each perhaps residing on a separate host. Doors are identified by the hostname of the server and the port number of the door service.

In a simple read-with-dCache operation, the client initiates a control connection with a dCache door, identified by its hostname and port number. The client requests a file from the cache via the control connection. The door authenticates the client and then passes the file request to its pool managers in round-robin fashion until it finds one which can deliver the file from its disk pools or it finds that none of its pool managers can do so. If the file is in a disk pool, the door transfers control to that pool's manager and a separate data connection is initiated. The datafile is then sent to the client via the data connection. If the file is not in any pool, the door finds a pool that has space for the file, and initiates a request to read that file from tape via Enstore into that pool. Then, too, a data connection between that pool's manager and the client is established and the file sent to the client.

Each dCache pool supports up to a fixed number of active transfers to Enstore (Stores), from Enstore (Restores), and to/from clients (Movers). Once a pool reaches an active limit for a kind of request, further requests are generally assigned to other pools. If all pools have reached their active limit for a kind of request, then further requests are queued for later execution by one of the pools.

dCache uses one unified namespace for all files, provided by the PNFS product [11]. PNFS looks like a standard Unix directory system. Each file entry is actually a meta-data record whose name is the same as the underlying datafile the record describes. PNFS can be thought of as a database that distributes meta-data via NFS, but does not distribute the underlying datafiles. Non-I/O operations on the PNFS directory system are operations on the meta-data records, not the underlying datafiles.

1.2 Accessing Data in DCache

dCache provides several means of accessing data, including a C-based client access library, a ROOT-based file class with dCache access, and a kerberized ftp program. The ROOT-based file class TDCacheFile is the means by which dCache use has been integrated into CDF DH software. Using TDCacheFile in an alternative DIM-DH interface implementation 'under the covers' allows the user to switch between using dCache or the CDF DIM/stager by adding or changing one line of TCL to a DHInputModule talk-to. TDCacheFile also provides the ability to access data in dCache from the interactive root prompt. Examples of these different approaches to data access are given in Section 2.1 and throughout Section 3.

1.3 Other Features and Planned Developments

The dCache design contains the concept of "attraction" or "affinity", where requests for certain file families (CDF datasets) or from certain network sub-nets will be forwarded to specific pools according to the dCache service configuration. This permits some optimization of the caching strategy. This feature has been applied, for example, to set aside a limited number of pools to exclusively cache raw data files, preventing raw data files, read once in a FIFO access pattern by the production farms, from eventually displacing all other files in cache. We could also apply this to optimize caching for CDF trailer PCs and the CAF. Requests coming from certain sub-nets could be directed to dCache pools on or 'near' those sub-nets to reduce network traffic.

dCache was designed for data access where network bandwidth is not a severe limitation. At DESY, for instance, dCache is used solely by on-site clients. Although dCache does support remote access and in fact is used now to serve both on-site and off-site SAM datafile requests, it just was not designed with remote client support as its fundamental goal. CDF consideration of dCache use includes support for data access by clients in the CDF trailers as well as in FCC, and perhaps even clients at remote institutions, where dCache will still be a valuable buffer between user analyses and tape-resident data. There are dCache development plans being pursued now at FNAL to address remaining remote (off-site) access issues using GRID-related transfer tools.

The dCache team has expressed interest in supporting the "operation on an island" concept. The goal is to permit limited use of dCache when network connectivity to the central CDF dCache system is lost. Read-only access to the files in accessible caches would be permitted under these conditions, with full access restored when network access was regained. This could be useful to permit some analyses on local files to continue during network outages, as well as to permit analyses in more exotic situations, such as on laptop computers while traveling. The development work for this feature is on-going.

2 Getting Started with DCache at CDF

There are two basic means of accessing datafiles via dCache at CDF. First, AC++ user analysis jobs may employ DHInputModule to access data in dCache as an alternative to using the DIM. This approach, which uses the ROOT class TDCacheFile, allows record-oriented access to single and multi-branched datafiles, and is described later in this section. Second, users may copy whole files out of dCache using the dCache 'dccp' copy program. This approach is described in Section 3.2 on page 10. There are variations on these approaches, as well as wholy different alternatives, described throughout this note.

The ability to use dCache transparently in place of the DIM in AC++ user analysis jobs has been possible since cdfsoft2 releases 4.6.2 and 4.6.0int3. Nowadays, to take advantage of many improvements in dCache client-server interaction, one must use cdfsoft2 release 4.8.5, 4.9.1hpt3, 4.10.1 or later, 4.11.X, 5.X, or development. Users are strongly encouraged to use cdfsoft2 release 4.11.2 or later, however, to improve overall CDF operations. This and subsequent releases contain improvements to database client software, for instance, that greatly simplify Offline database operations. Any Linux or IRIX computer with a qualifying release can be used. No specific authentication mechanism is required, though that will change when write capabilities become public.

The stage 2 CAF file servers and the CAF static file servers [1] are now in dCache service. The current read cache capacity is 93 TB, 50 TB which now functions as a semi-static "Golden Pond" read cache, 26 TB of which is an unsegmented general read cache, and the remaining space consists of specialized raw data and large dataset caches. Compare this to the 10-15 TB of space in the DIM.

There have been two distinct Data Handling-aware input modules: the original 'old' DHInputModule [12] and the next generation 'new' DHInputModule. The new DHInputModule has been the default for quite some time now, so only it is now described. Older versions of this note can be consulted on the use of the OldDHInput with dCache. For the sake of simplicity, this note refers to the new DHInputModule simply as DHInput, its short-hand module name.

2.1 Example of Using DCache with DHInputModule

As a first example, let's use AC++Dump on a central machine (fcdfsgi2, fcdflnx2, fcdflnx3) with cdfsoft2 4.11.2. One line of TCL selects the cache system used by DHInputModule: "cache set MYCHOICE", where MYCHOICE is one of:

- DIM or KAHUNA CDF DIM is used for caching data files.
- DCACHE or DCAP dCache (no PNFS mode) is used for caching data files.
- DCACHE-PNFS dCache (PNFS mode) is used for caching data files.
- SRM dCache (no PNFS mode) with GRID SRM front-end.

The default choice in 4.11.2 is DCACHE [1]. The two dCache access modes have slightly different characteristics. The DCACHE mode is our current recommended choice since it does not require a PNFS mount on the client machine. The DCACHE-PNFS mode requires a PNFS mount, but can also be used for writing. Both modes work on our central platforms, which have PNFS mounted, but the DCACHE mode alone will work elsewhere. Note that permission to NFS mount PNFS is granted only on an as-needed basis and requires CDF Computing management approval. In general, CDF computers will not NFS mount PNFS. The SRM dCache more uses a GRID tool as a front-end to dCache to allow more control over dCache access on the server side. It is our long-term choice for dCache access, but currently is still in beta test level of maturity.

The file Edm/tools/DhAccessDCache.tcl in the CDF Offline repository contains examples of DHInput instructions which read datafiles using the DIM or dCache, using either dataset, fileset, or file specification. The following example in Figure 1 on page 5 demonstrates using AC++Dump driven by Edm/tools/DhAccessDCache.tcl. This and the ensuing examples have been tested extensively on many systems, including fcdfsgi2 and the CAF [13], using the same TCL.

```
% setup cdfsoft2 4.11.2
% # Excerpt from Edm/tools/DhAccessDCache.tcl
module input DHInput
  talk DHInput
    include dataset  gexo0b  book=cdfpexo
    cache set DCACHE
  exit
begin -nev 1010

% AC++Dump Edm/tools/DhAccessDCache.tcl
<SNIP>
*****
***** Opened input file: dcap://cdfdca.fnal.gov:25125/pnfs/fnal.gov/
*****     usr/cdfen/filesets/GI/GI05/GI0500/GI0500.0/gb01defd.0001exo0
*****
DHInput Begin processing 100th record. Run 138425, Trigger 37617
DHInput Begin processing 200th record. Run 138425, Trigger 80857
<SNIP>
```

Figure 1: Example of using AC++Dump with dCache

[1] The default choice in some older versons is DIM.

2.2 Using DCache from the CAF

dCache can be used by CAF jobs exactly as it is used by AC++ jobs on one's desktop. Use the same TCL to select the cache mechanism as shown in the examples. The only access method that will work on the CAF is DCACHE. PNFS is not mounted on the CAF worker nodes, so DCACHE-PNFS will not work. For details on how to best input large datasets split across many sections, refer to the CAF Users Manual.[13]

2.3 Monitoring DCache

The CDF dCache system is generically referred to as CDFDCA. In the current CDFDCA configuration there are three administrative nodes (cdfdca, cdfdca2, and cdfdca3) which do not host any pools (they could, but we chose for them not to). There are many general-purpose read pools hosted by terabyte file-servers, for instance r-dataNNN-K, where NNN = 017,018,019,020 and K = 1,2,3. There are also a few write pools hosted by terabyte file-servers, such as w-dataNNN-1, where NNN = 008,009. This configuration is likely to expand and change over time. Users still interact with the dCache the same way though.

The main dCache monitoring web page can be found at http://cdfdca.fnal.gov. A variety of plots are available from this page to track dCache usage by activity, by number of files, and by number of bytes. To view current dCache activity, select 'DCA System Status'. The Pool Usage page shows the space usage in dCache pools. The Pool Request Queues page shows the number of active file Moves, Restores, and Stores in the system. The Active Transfers pages contain a table of detailed information on all active client transactions. Most dCache monitoring pages do not update automatically and do experience some time lag, so refresh them every so often.

Enstore's home page, http://www-cdfen.fnal.gov:/enstore/enstore_system.html, may be useful when Stores and Restores are in progress. I find the 'Enstore Server Status' page most useful to monitor tape access by dCache. Service requests from dCache have username 'root' while the hostname is one of fcdfdataNNN.

Finally, the CAF project has a number of monitoring tools which can track CPU and network useage by the CDFDCA pool hosts. These can be found linked to by http://cdfcaf.fnal.gov.

2.4 Pre-staging in AC++

If you select dCache caching in AC++, all files within the first and second filesets in your input request will be 'staged' from tape to disk at the beginning of the job. If the files are already on disk (in a pool in dCache), staging will be a null operation. If the files are on tape, then actual staging begins. The number of Enstore requests that dCache will make is limited by the number of active requests serviced by our pools, which is a CDFDCA configuration parameter that will be tuned as more pool hosts arrive. Requests for data on tape will be initially listed as Restore requests while the files are fetched from Enstore (if the files are not already in cache), then as Mover requests while the files are sent to the client. As the end of each fileset is reached by DHInput, the fileset after the next will be staged in order to insure that filesets are generally on disk in the cache before the input module opens them for reading.

2.5 DH_DCacheCheck

DH_DCacheCheck is a CDF-developed utility which permits users to check whether portions of CDF datasets, filesets, or individual files are on disk in dCache, and optionally to copy that data to a local disk. It is effectively a wrapper on the dCache utilities dccp, dc_check, and dc_stage, a wrapper that understands and translates CDF meta-data concepts like datasets and filesets. Figure 2 on page 8 shows how it may be used at present to check on the status of one file in the cdfpexo book, one fileset in the cdfpexo book, and 10 filesets in reverse order from a dataset in the cdfpexo book. Executing 'DH_DCacheCheck' with no arguments will list what command line options it supports. DH_DCacheCheck is still being extended to meet user needs, so be sure to check the help listing (-h) for the latest supported functions of the program.

2.6 Known Differences and Limitations

The current implementation of the AC++ dCache interface does now re-order files within a fileset according to what is on disk in cache already and what is not. Whole filesets are still staged in the order presented to the dCache interface. Within any one fileset, however, files in cache are delivered to the user before files not in cache. The has proven very useful for cases where a few files from a fileset must be restored from tape, and the user program can proceed to process events from disk file those files are fetched.

The ability to disable 'input launch' with the DIM to force jobs to only read

```
% setup cdfsoft2 4.10.2
% DH_DCacheCheck -F gb01defd.0001exo0 -b cdfpexo
File: gb01defd.0001exo0 IN-CACHE.
--------------------------------------------------------------------------------
Summary: File total:   1 files:  1 in-cache,  0 to-stage,  0 uncached,  0 error
Summary: Fset total:   0 filesets


% DH_DCacheCheck -f GI0500.0 -b cdfpexo
Fileset: GI0500.0 has  9 files:  4 in-cache,  0 to-stage,  5 uncached,  0 error
--------------------------------------------------------------------------------
Summary: File total:   9 files:  4 in-cache,  0 to-stage,  5 uncached,  0 error
Summary: Fset total:   1 filesets


% DH_DCacheCheck -d gexo0b -b cdfpexo -n 10 -r
Dataset: gexo0b has   5 filesets
--------------------------------------------------------------------------------
Fileset: GI0503.0 has  7 files:  0 in-cache,  0 to-stage,  7 uncached,  0 error
Fileset: GI0500.3 has  9 files:  0 in-cache,  0 to-stage,  9 uncached,  0 error
Fileset: GI0500.2 has  9 files:  0 in-cache,  0 to-stage,  9 uncached,  0 error
Fileset: GI0500.1 has  9 files:  0 in-cache,  0 to-stage,  9 uncached,  0 error
Fileset: GI0500.0 has  9 files:  4 in-cache,  0 to-stage,  5 uncached,  0 error
--------------------------------------------------------------------------------
Summary: File total:  43 files:  4 in-cache,  0 to-stage, 39 uncached,  0 error
Summary: Fset total:   5 filesets
```

Figure 2: Example of using DH_DCacheCheck

data on disk and ignore data only on tape has not yet been implemented with dCache. This is technically feasible, but will only be implemented if a case is made for its utility with dCache.

Load balancing in general (20 requests for files not yet in cache) does work amongst the pools, but the special case of file replication (20 requests for files on disk in one pool) is being worked on. The feature exists, but has not yet been tuned yet to operate as desired. Presently, all 20 requests will go to the pool already hosting the data files, perhaps causing some requests to queue up. This is not really a functional defect since all requests are served eventually, but it can be a performance bottleneck.

Strong authentication issues with regards to dCache data access at Fermilab have been resolved. Kerberized dCache access is now in testing. dCache write access deployment depends on this resolution. In the meantime, write pools in dCache are being monitored and tested internally by CCF department and the CDF DH group. Kerberized access also will help uniquely identify all dCache clients, including CAF clients and off-site clients.

3 More Examples of Accessing Data via DCache

There are many ways to access data with dCache, some of which are demonstrated here. CDF users are not likely to find all of these examples useful in practice, however, since most users will simply use dCache with the DHInput module as described in Section 2.1. For the curious and the adventurous, however, here are more ways to exploit dCache to access CDF datafiles.

3.1 dCache Access Protocol

Several means of accessing data in dCache use the dCache Access Protocol library, dcap. Dcap is a dCache-aware implementation of basic POSIX file functionality written in C. It provides the functionality required to read and write whole files or parts of files, as well as to seek back and forth within files. There are several different approaches to applying dcap, depending on whether one has PNFS mounted on the client host or not, and whether one uses a ROOT class TDCacheFile as a jacket on dcap. The example described in Section 2, for example, uses TDCacheFile without PNFS. To use dcap to access files in dCache, one needs to install the dcap product on the client host. The dcap version which is part of the modern cdfsoft version 4.10.2 configuration is v2_24_f0203.

Many of the examples below use the utility programs supplied by dcap: dccp, dc_stage, and dc_check. All examples were performed on fcdflnx2, but have been proven to work elsewhere as well. PNFS use is limited to machines with PNFS mounted, of course. The minimal product setup is shown for each example, but all will work with 'setup cdfsoft2 4.11.2'.

3.2 Client Access with Dcap

World-readable files in dCache can be accessed without mounting PNFS on the client using a URL-like 'dcap:' syntax to specify the file. This approach also works if PNFS is mounted. The following example in Figure 3 on page 10 demonstrates using dccp to copy a file from dCache to a local disk using the 'dcap:' syntax. The filepath specification has been edited to span two lines because of its length. It uses the 'global' PNFS file specification. The port number is specified here to indicate which dCache door on the admin nodes to use.

```
% setup dcap v2_25_f0519
% set adminhost  = "cdfdca.fnal.gov"
% set doorport   = "25125"
% set globalpnfs = "/pnfs/fnal.gov/usr/cdfen/filesets"
% set fileset    = "GI/GI05/GI0500/GI0500.0"
% set filename   = "gb01defd.0001exo0"
% set localdisk  = "/cdf/scratch/kennedy"

% dccp dcap://$adminhost':'$doorport$globalpnfs/$fileset/$filename \
              $localdisk/$filename
1024552576 bytes in 89 seconds (11242.02 KB/sec)

% ls -alFG $localdisk/$filename
-rw-r--r--    1 kennedy  1024552576 Nov 23 11:53 gb01defd.0001exo0
```

Figure 3: Example of using dccp without PNFS

The most general means to access data using dcap is with PNFS mounted on the client. PNFS access is required to write into dCache with dcap for instance, and it is required to read files that are not marked world-readable. Permission to NFS mount PNFS is granted only on an as-needed basis and requires CDF Computing management approval. In general, CDF computers will not NFS mount PNFS. CDF central machines, however, do have PNFS mounted.

Figure 4 on page 11 shows dccp being used to copy a file to a local disk exploiting the PNFS mount on the client host. Note that unlike other access modes, a port number is not specified here since it is obtained 'under the table' from a configuration parameter stored in PNFS.

```
% setup dcap v2_25_f0519
% set localpnfs = "/pnfs/cdfen/filesets"
% set fileset   = "GI/GI05/GI0500/GI0500.0"
% set filename  = "gb01defd.0001exo0"
% set localdisk = "/cdf/scratch/kennedy"

% dccp $localpnfs/$fileset/$filename $localdisk/$filename
1024552576 bytes in 91 seconds (10994.94 KB/sec)

% ls -alFG $localdisk/$filename
-rw-r--r--    1 kennedy  1024552576 Nov 23 12:08 gb01defd.0001exo0
```

Figure 4: Example of using dccp with PNFS

3.3 Client Access with TDCacheFile

TDCacheFile is a dCache-aware ROOT TFile class written by Grzegorz Mazur of DESY. TDCacheFile is distributed with the ROOT package as of ROOT version 3.03, and is actively supported by the author. TDCacheFile adapts the ROOT TFile class interface to transparently operate on files in dCache, using dcap for implementation. TDCacheFile allows data in dCache to be treated by any of the usual means available in the ROOT TFile universe. Data can be accessed by files, trees, branches, objects, and so on. The class has the same access modes as dcap with respect to PNFS mounts. It permits the reading of world-readable files without PNFS. To write any data into dCache or read non-world-readable files, a PNFS mount is required on the client host.

TDCacheFile exploits ROOT v3's TPlugInModule apparatus to be a child class of TFile without the TFile factory actually having to know about it. This requires a little bit of setup code to be executed along with ROOT global initialization. The TDCacheFile plug-in module setup triggers the dynamic loading of libDCache.so if and only if TFile sees files specified with the prefixes 'dcap:' or 'dcache:'. The former prefix is the previously mentioned 'dcap:' syntax enabling access without a PNFS mount. The latter prefix is the syntax for used for access exploiting a PNFS mount. This ROOT configuration setup is automatically performed in the Framework and EdmUtilities packages, if one compiles them against ROOT v3.03 or later.

Note that TDCacheFile works with – *any* – ROOT datafile, whether it contains EDM-format data or ROOT ntuples, single-branch or multi-branch. With TD-CacheFile, we can read tape-resident ntuple files, making Enstore's features available for large-scale ntuple-based analysis. This does not mean that ntuples will be read and analyzed in AC++ in the same manner as EDM-format data, however. Ntuples are analyzed exactly the same as before; only transparent tape access is added by TDCacheFile. Some means of using meta-data to drive ntuple-based analyses may be useful to fully exploit this feature.

Three examples of accessing CDF datafiles are shown below. First, Figure 5 on page 12 shows a listing of the objects in one event record from a file using dCache using Edm_ObjectLister. The program output is slightly edited to remove extraneous details.

```
% setup cdfsoft2 4.11.2
% set localpnfs = "/pnfs/cdfen/filesets"
% set fileset   = "GI/GI05/GI0500/GI0500.0"
% set filename  = "gb01defd.0001exo0"

% Edm_ObjectLister -n 1 dcache:$localpnfs/$fileset/$filename

  Entry#     Run# Section# Trigger# Record Type          Nbytes In
-------- -------- -------- -------- -------------------- ---------
       0   122621        1       33 EmptyRunSections           150

ByteIn  Class Name              (ObjectId:BankInfo|RCPID) Proc Name
------  ----------------------- ------------------------- ---------
   126  LRIH_StorableBank       (       1: LRIH,    0, 0)
===================================================================
```

Figure 5: Example of using Edm_ObjectLister with dCache

The second example demonstrates the reading of just one ROOT branch stored in a large datafile using Edm_EventLister. The 'info' branch that is read by this program contains event information used to implement random event access by the Edm and DHMods packages. That information occupies very little space compared to the event data in the main 'data' branch. The time it takes to read the small info branch is much less than the time it takes to read the entire datafile, demonstrating that TDCacheFile reads and transfers over the network just the one small branch data to the client. The output, presented in Figure 6 on page 13, has been edited to remove extraneous details and to reduce its size.

Finally, TDCacheFile can be utilized from the root prompt to access files, whether Edm data files or plain ntuples. Figure 7 on page 14 demonstrates using the root interpreter to open and print TTree information from a CDF datafile. The same approach could be used to process ntuples with dCache. One could put the

```
% setup cdfsoft2 4.11.2
% set localpnfs = "/pnfs/cdfen/filesets"
% set fileset   = "GI/GI05/GI0500/GI0500.0"
% set filename  = "gb01defd.0001exo0"

% Edm_EventLister dcache:$localpnfs/$fileset/$filename

dcache:/pnfs/cdfen/filesets/GI/GI05/GI0500/GI0500.0/gb01defd.0001exo0

NaturIdx EntryIdx     Run# Section# Trigger# Record Type
-------- -------- -------- -------- -------- --------------------
       0        0   122621        1       33 EmptyRunSections
       1        1   122621       43       46 EmptyRunSections
       2        2   122621       54       95 EmptyRunSections
     ...      ...      ...      ...      ... ...
   13687    13686   123560       85   309692 PhysicsEvent
   13688    13687   123560       85   309768 PhysicsEvent
```

Figure 6: Example of using Edm_EventLister with dCache

PlugInManager setup commands shown in the figure into one's .rootrc file to avoid retyping them in each root session. The output shown is heavily edited, but shows some of the basics of the ROOT interaction. Note that the precise syntax of the AddHandler calls in the example have varied in past versions of ROOT. See the file Edm/src/initRootPackage.cc for an example of the older syntax.

3.4 Client Access Optimization

Dcap also contains methods to help optimize client access to data. The method dc_check() returns true or false depending on whether a file is on disk in a dCache. This indicates the 'cost' of accessing the data, as tape access is a more limited resource. The method does not indicate which pool in the cache that the file might reside in, only whether it is in any pool of the dCache. For use by scripts, this method is available as the program dc_check in the dcap package. In the future, dc_check() might be used by DH software to re-order files requested by clients, to deliver 'cheap' files in cache first, files on tape last, or to avoid files on tape altogether.

The method dc_stage() stages a file from tape to disk without delivering it to a client. The method does not guarantee the arrival time of the file in cache. For use

```
% setup root v3_05_04d -q KCC_4_0
% setup dcap v2_25_f0519
% root
root [0] gROOT->GetPluginManager()->AddHandler("TFile", "^dcache:",
                "TDCacheFile", "DCache", "TDCacheFile(const char *)) ;
root [1] gROOT->GetPluginManager()->AddHandler("TFile", "^dcap:",
                "TDCacheFile", "DCache" "TDCacheFile(const char *)) ;
root [2] TFile*p_file = TFile::Open("dcache:/pnfs/cdfen/filesets/
                GI/GI05/GI0500/GI0500.0/gb01defd.0001exo0") ;
root [3] TTree* p_tree = p_file->Get("Sequential") ;
root [4] p_tree->Print() ;
******************************************************************************
*Tree    :Sequential: CDF Root Event                                         *
*Entries :    13689 : Total =      1904042052 bytes  File  Size = 1024269775 *
*        :          : Tree compression factor =   1.86                       *
******************************************************************************
*Br    0 :Sequential :                                                       *
*Entries :    13689 : Total  Size= 1903314257 bytes  File Size  = 1023541980 *
*Baskets :    13538 : Basket Size=     131072 bytes  Compression=   1.86     *
*............................................................................*
*Br    1 :EventInfo :                                                        *
*Entries :    13689 : Total  Size=     612376 bytes  File Size  =     612376 *
*Baskets :      164 : Basket Size=       8192 bytes  Compression=   1.00     *
*............................................................................*
```

Figure 7: Example of using the root interpreter to access files in dCache

by scripts, this method is available as the program dc_stage in the dcap package. The dc_stage() method is now used by DH software to stage files to cache before they are opened by a client, when possible.

In the examples shown in Figure 8 on page 15, the use of dc_check and dc_stage is demonstrated. The ‘gb01defd.0001exo0’ datafile is initially in the disk cache, but the files ‘NotInCache.OrPnfs’ and ‘gb01e2a8.0056exo0’ are not. The first three dc_check calls confirm this. Then the latter datafile is staged using dc_stage to the disk cache from tape, without being delivered to a client. Finally, the last dc_check confirms the datafile is indeed in the disk cache. ‘Check passed’ means the file is in cache, and ‘Check FAILED’ means the file is not in cache. Note that the text output here has been edited to reduce its size.

```
% setup dcap v2_25_f0519
% set localpnfs = "/pnfs/cdfen/filesets"
% set fileset   = "GI/GI05/GI0500/GI0500.0"

% dc_check $localpnfs/$fileset/gb01defd.0001exo0
Check passed for file...

% dc_check $localpnfs/$fileset/NotInCache.OrPnfs
Check FAILED for file...

% dc_check $localpnfs/$fileset/gb01e2a8.0056exo0
Check FAILED for file...

% dc_stage $localpnfs/$fileset/gb01e2a8.0056exo0 90
Stage passed for file...

% dc_check $localpnfs/$fileset/gb01e2a8.0056exo0
Check passed for file...
```

Figure 8: Example of using dc_check and dc_stage

3.5 Client Access with Kftp

A more exotic means of accessing data in dCache is the 'kftp' product, a kerberized ftp-like suite of programs which can communicate with a dCache server. Like ftp, kftp transfers data in units of files. Use of kftp does not require the mounting of PNFS on the client machine, but does require registration of the client's kerberos principal with the central dCache administrator. kftp is presented here because its role is similar to that of a dCache-aware GRIDftp, the development of which is being planned at FNAL. Such a tool should improve dCache access from remote institutions.

The kftp product delivers two programs: 'kftpls' to perform a file listing, and 'kftpcp' to perform a file transfer. These programs are implemented and distributed as interpreted Python modules. To prepare to use kftp, a user must register one's Kerberos principal with enstore-admin@fnal.gov to be allowed access. Next, install and setup the kftp product, along with Python and other dependencies. I used kftp v1_0 on a Linux desktop PC in the CDF trailers running FRH 6.1 for this example.

Figure 9 on page 16 contains an example of using kftpls and kftpcp. First, kftpls is used to create a file listing of the virtual tape labels being used by CDF at present: CA, GI, and PR. Then, kftpcp is used to copy a file from dCache to a local disk.

```
% setup kftp v1_0
% set adminhost = "cdfdca.fnal.gov"
% set doorport  = "25127"
% set localpnfs = "/filesets"
% set fileset   = "GI/GI05/GI0500/GI0500.0"
% set filename  = "gb01defd.0001exo0"
% set localdisk = "/data3/kennedy"

% kftpls -p $doorport -m p kennedy@$adminhost':'$localpnfs/
CA
GI
PR

% kftpcp -p $doorport -m p \
    kennedy@$adminhost':'$localpnfs/$fileset/$filename \
    $localdisk/$filename

% ls -alFG $localdisk/$filename
-rw-r--r--   1 kennedy  1024552576 Nov 23 13:09 gb01defd.0001exo0
```

Figure 9: Example of using kftpls and kftpcp

4 Conclusion

dCache has been demonstrated to be a fully capable distributed datafile caching system, providing numerous means to access CDF data stored on tape in Enstore. The DHInputModules have been adapted to provide a choice of using the existing CDF DIM system or either of two modes to access data via the CDFDCA dCache system. The CDFDCA system is nsoon to reach 100 TB capacity, and cdfsoft2 software releases which can use dCache are now widely available. As of just recently, the dCache system is considered to be in production status, capable of handling the data access demands of CDF.

References

[1] Robert M. Harris, et al., "CDF Plan and Budget for Computing in Run 2", CDF Document CDF/DOC/COMP_UPG/PUBLIC/5914, 16 May 2002, Version 3

[2] Paul Hubbard and Stephan Lammel, "The CDF Run II Disk Inventory Manager", CHEP 2001, September 2001, http://www-cdf.fnal.gov/upgrades/cdfdh/doc/chep2001/4-035.pdf

[3] Paul Hubbard, "DIM/Kahuna Design Write-up", CDF Document CDF/DOC/COMP_UPG/PUBLIC/5312, April 2000

[4] Paul Hubbard, "Maintaining the Kahuna and Stager', TeX file found in the doc sub-directory of the stager product

[5] Robert D. Kennedy, et al., "Technical Review of CDF Data Handling Infrastructure Software", CDF Document CDF/DOC/COMP_UPG/PUBLIC/5917, 3 June 2002, Version 1.01

[6] http://www-dcache.desy.de

[7] "dCache, a distributed Data Caching System", draft white paper, http://www-dcache.desy.de/dCacheDesign.html

[8] Patrick Fuhrmann, et al., "A Distributed Rate-Adapting Buffer Cache for Mass Storage Systems", CHEP 2000, February 2000

[9] Patrick Fuhrmann, "dCache", HEPIX 2000, April 2000

[10] Michael Ernst, et al., "dCache, a distributed storage data caching system", CHEP 2001, September 2001, http://www-dcache.desy.de/chep2001/talk-4-005.pdf

[11] Patrick Fuhrmann, "A Perfectly Normal FileSystem", CHEP97, Spring 1997, http://watphrakeo.desy.de/pnfs

[12] F. Ratnikov, "Input and Output Modules User Guide", CDF Document CDF/DOC/COMP_UPG/PUBLIC/5336, 08 June 2000, http://rutpc7.fnal.gov/ratnikov/Docs/DHInputModuleReference.htm

[13] M. Casarsa, et al., "CDF CAF Users's Manual", CDF Document CDF/DOC/COMP_UPG/PUBLIC/6092, 21 August 2002, http://cdfcaf.fnal.gov/doc/UserGuide.ps

[14] E. Wicklund, private communication, 12 November 2002.