# A caching DPM

## Volatile Pools

Oliver Keeble on behalf of the DPM team

# The client view

- The volatile area is accessed by path
  - /dpm/cern.ch/home/dteam/volatile
- Reads from a volatile pool work normally if the file is there
- If the file is not there, it is retrieved
  - The FULL FILE is retrieved (no chunks)
  - The client will block while this happens
- Writes into a volatile pool go into the volatile pool (if there is space)

# Volatile pools and caches in a nutshell

- This functionality requires a working DOME installation
- Marking a pool as “Volatile” triggers the cache-like behaviour for that pool.
  - Others stay like before
- This creates a data cache that works seamlessly and interchangeably with all the data protocols: **HTTP, Xrootd, GridFTP**
  - Not supported for SRM

# Volatile pools and caches in a nutshell

- If the requested file is **not** there, DPM will try to fetch it from an external source
  - Two scripts are required
    - one does stat()
    - the other one retrieves the new file
  - If there's no space in the pool, DPM will remove some files from it before invoking the pull script. It's volatile!
    - The oldest file on the fs selected for write will be removed
      - No access time awareness

# Volatile pools and caches in a nutshell

- The file retrieval activity is properly queued and scheduled. A GET storm will behave in a coordinated manner
  - No more than N retrievals per server
  - No more than M retrievals overall
  - Clients peacefully wait their turn, it’s transparent
- Space reporting will work as usual

# What can be interfaced

- A DPM volatile pool can cache virtually any source of files, remote or not
- We provide two simple example scripts that fake an external source, just to show the parameters
  - These are called **only if the file is not resident**
- Who does the integration will have to adapt these two scripts or executables to his own external source
- This script will probably need credentials of some kind to do its work

```
Head $> ls -l /usr/share/dmlite/filepull
-rwxr-xr-x. 1 root root 303 Apr 19 17:16 externalstat_example.sh
```

```
Disk $> ls -l /usr/share/dmlite/filepull
-rwxr-xr-x. 1 root root 587 Apr 19 17:16 externalpull_example.sh#
```

# Setup

- Create a volatile pool and add filesystems
  - Ensure pool default filesize > largest file you will cache
- Create a QT on that pool
- Assign the QT to your /volatile path
- Configure the stat script on the head node
- Configure the pull script on the disk servers
- See the DPM deployment guide for more
  - https://twiki.cern.ch/twiki/bin/view/DPM/DpmSetupDpmCache
- And the DOME reference for all the details
  - http://lcgdm.web.cern.ch/dome-documentation

# Setup

```
dmlite-shell> pooladd VolPool filesystem V
dmlite-shell> poolmodify VolPool defsize 4000000000
dmlite-shell> fsadd …
dmlite-shell> quotatokenset /my/volatile/path …
```

```
# grep filepuller /etc/domehead.conf
head.filepuller.stathook:
/usr/share/dmlite/filepull/externalstat_example.sh
```

```
# grep filepuller /etc/domedisk.conf
disk.filepuller.pullhook:
/usr/share/dmlite/filepull/externalpull_example.sh
```

- Other config options are available, for example controlling the number of concurrent pulls
  - https://twiki.cern.ch/twiki/bin/view/DPM/DpmSetupDpmCache

# Example stat script

- One scripts to perform stat() towards the remote endpoint
- Returns filesize/checksum or error if not found

```
#!/bin/sh

# usage: externalstat.sh <lfn>

# This is an example script for the DOME file pull hooks
# This script will make DOME believe that there is an external file
# that has 123456 as its size
# The companion pulling script will have to create such a file when
invoked

echo ">>>>> STAT 123456"
```

# Example pull script

- One script to pull the file. Ret: OK or error

```
#!/bin/sh

# usage: externalpull.sh <lfn> <pfn>

# This is an example file puller script, that creates a fake file <pfn>
# by pulling it from nowhere using dd
# If querying an external system, the query should be based on the <lfn>
#

# Let's claim we're doing something complex and heavy
sleep 5

# Pull the file
dd "of=$2" "if=/dev/urandom" bs=123456 count=1
```

# With the default setup you can…

```
# start empty
> gfal-ls davs://dpm/volatile

# see a file that isn't there (stat script is triggered)
> gfal-ls -l davs://dpm/volatile/not_there

# get a file (pull script is triggered)
> gfal-cat davs://dpm/volatile/tf01

# see it in the namespace
> gfal-ls -l davs://dpm/volatile/tf01

# remove it
> gfal-rm davs://dpm/volatile/tf01

# notice that it's "still there" (stat script is triggered)
> gfal-ls -l davs://dpm/volatile/tf01

# upload a file (--just-copy to avoid an initial stat)
> gfal-copy -just-copy file:///etc/services davs://dpm/volatile/tf02
```

# Scenarios

- Cache + primary storage
  - A satellite site can accelerate access to a nearby custodial storage system
  - This could allow a group of nearby sites to consolidate their storage
- Cached access to a federation
  - The upstream server can in fact be a federator such as Dynafed
  - This would transparently accelerate access to a federation
- Federating the cache
  - Possible – depends on the desired behaviour

# Possible extensions

- Client blocking
  - We could redirect clients to the origin rather than blocking them
    - Lower latency, more WAN traffic
- Federating the cache
  - A federator always sees the cache as full
    - If it redirects a client there, the pull is triggered
  - We could implement a way to query cache residency to make this more flexible

# Summary

- A contribution to the “caching conversation” happening in WLCG now.
- We are interested in understanding more about what caching scenarios suit smaller sites
  - Please let us know your experience
- More details in the two talks by Alessandra Doria, Silvio Pardi and Davide Michelino