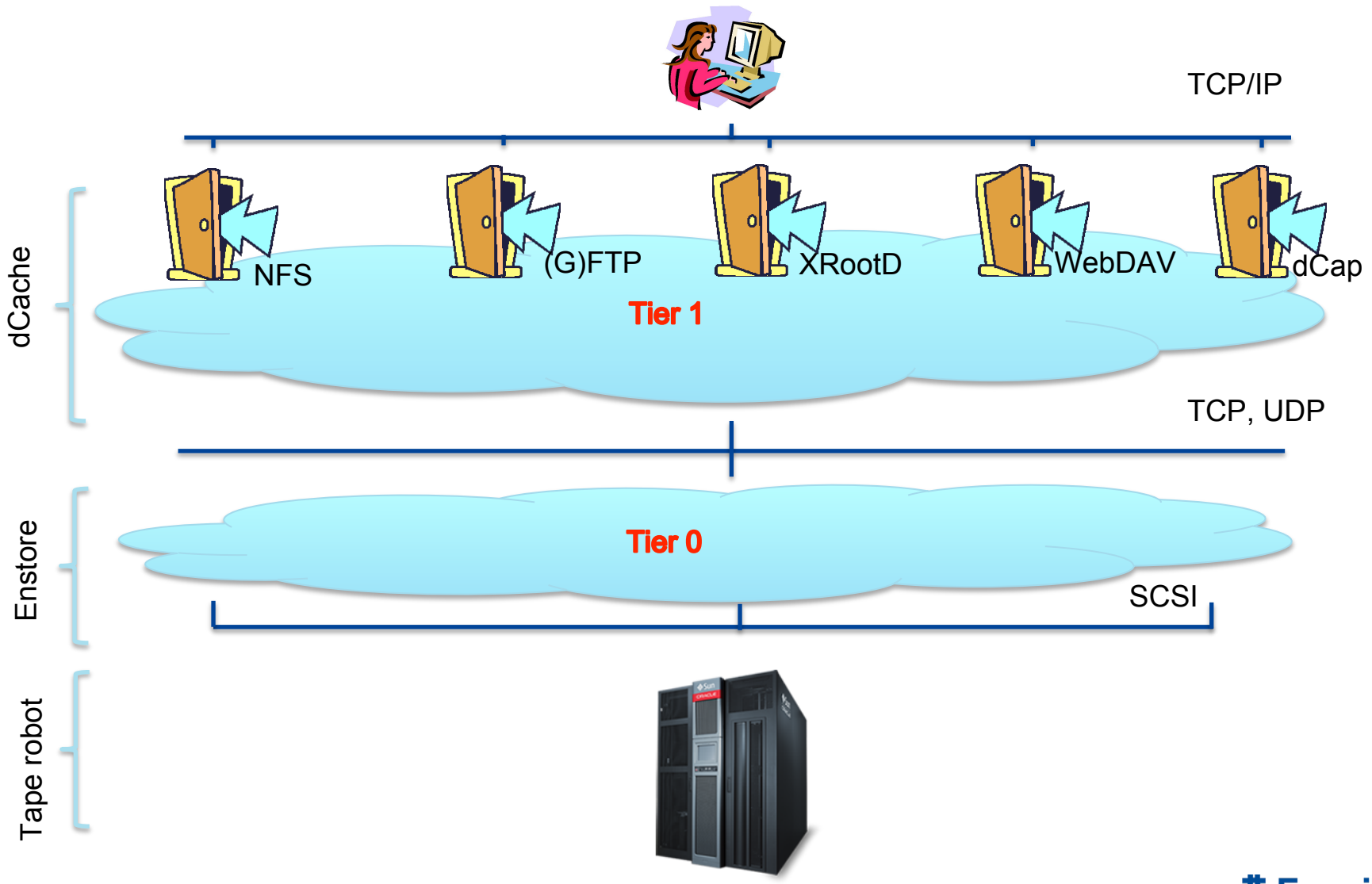
Best Practices: dCache

Dmitry Litvintsev

FIFE Workshop

20th-21st June 2016

dCache: Tiered data access



Public dCache

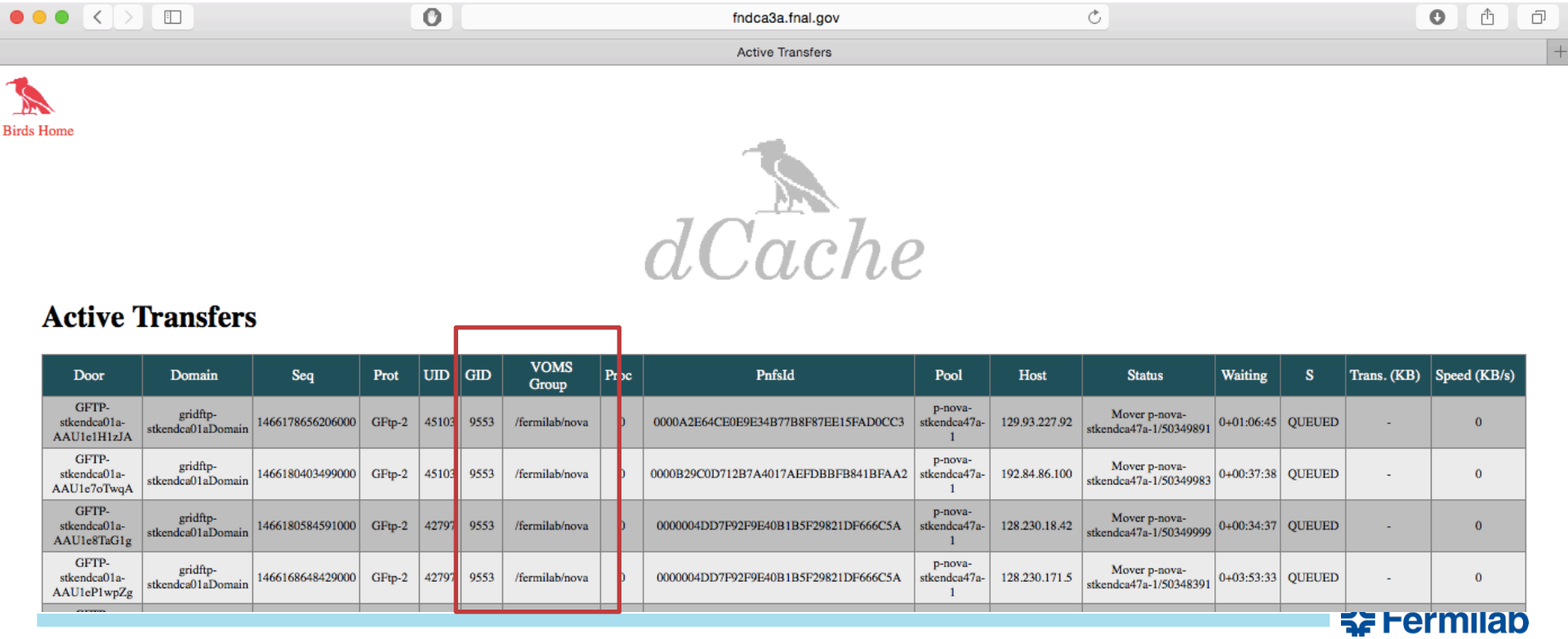
- Public dCache instance currently has about 7PB of disk space.
  - It provides scalable non-POSIX and POSIX-like access to Scientific data stored on immutable files distributed across multiple data servers.
- Recent changes (details in next 3 slides):
  - Upgraded from 2.6 to 2.12 (2015-08-20, CHG000000009656, 8 hours).
  - Tripled SSD capacity for dCache namespace, 400GB -> 1.2 TB (2015-10-06, CHG000000010110, 3 hours).
  - Tripled SSD capacity for dCache namespace, 1.2TB -> 3.5 TB (2016-05-19, CHG000000011217, 4 hours.
  - Upgraded from 2.12 to 2.13 (2016-06-16, CHG000000011360, 2 hours).

What's new in 2.13

- Not many user visible changed. A lot of improvements, particularly in NFS and bug fixes.
- Now we publish VO information about file transfers

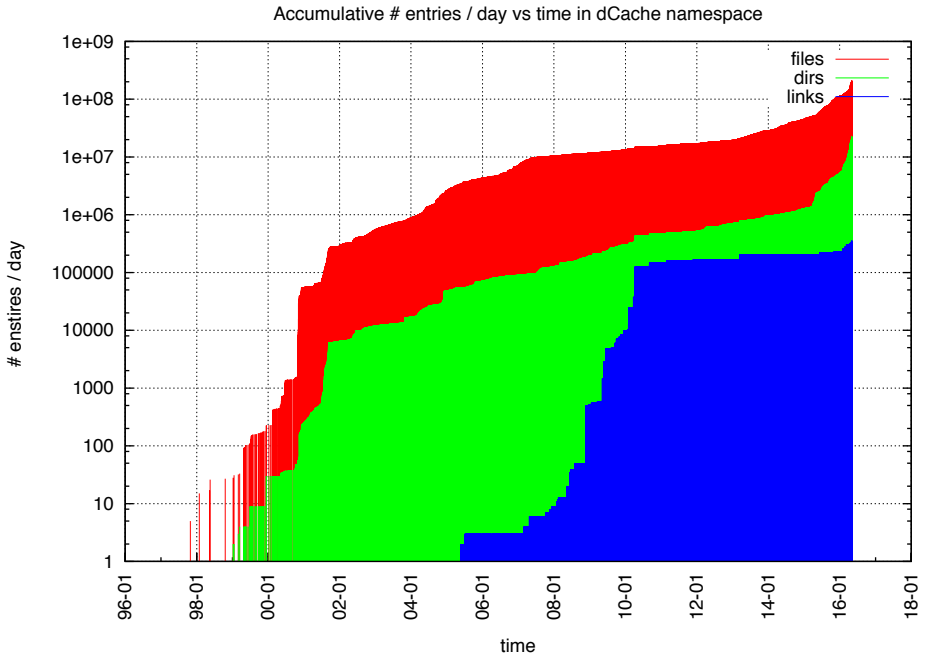
http://fndca3a.fnal.gov:2288/context/transfers.json

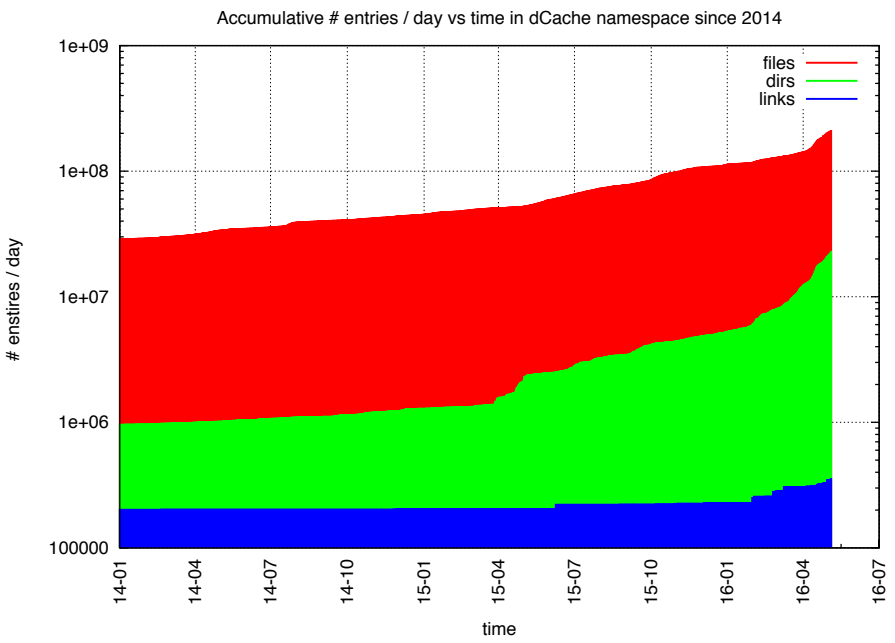
http://fndca3a.fnal.gov:2288/context/transfers.html



Door	Domain	Seq	Prot	UID	GID	VOMS Group	Proc	PnfsId	Pool	Host	Status	Waiting	S	Trans. (KB)	Speed (KB/s)
GFTP-stkendca01a-AAU1e1H1zJA	gridftp-stkendca01aDomain	1466178656206000	GFtp-2	45103	9553	/fermilab/nova	0	0000A2E64CE0E9E34B77B8F87EE15FAD0CC3	p-nova-stkendca47a-1	129.93.227.92	Mover p-nova-stkendca47a-1/50349891	0+01:06:45	QUEUED	-	0
GFTP-stkendca01a-AAU1e7oTwqA	gridftp-stkendca01aDomain	1466180403499000	GFtp-2	45103	9553	/fermilab/nova	0	0000B29C0D712B7A4017AEFDBBFB841BFAA2	p-nova-stkendca47a-1	192.84.86.100	Mover p-nova-stkendca47a-1/50349983	0+00:37:38	QUEUED	-	0
GFTP-stkendca01a-AAU1e8TaG1g	gridftp-stkendca01aDomain	1466180584591000	GFtp-2	42797	9553	/fermilab/nova	0	0000004DD7F92F9E40B1B5F29821DF666C5A	p-nova-stkendca47a-1	128.230.18.42	Mover p-nova-stkendca47a-1/50349999	0+00:34:37	QUEUED	-	0
GFTP-stkendca01a-AAU1eP1wpZg	gridftp-stkendca01aDomain	1466168648429000	GFtp-2	42797	9553	/fermilab/nova	0	0000004DD7F92F9E40B1B5F29821DF666C5A	p-nova-stkendca47a-1	128.230.171.5	Mover p-nova-stkendca47a-1/50348391	0+03:53:33	QUEUED	-	0

Public dCache: what's up with ever growing DB



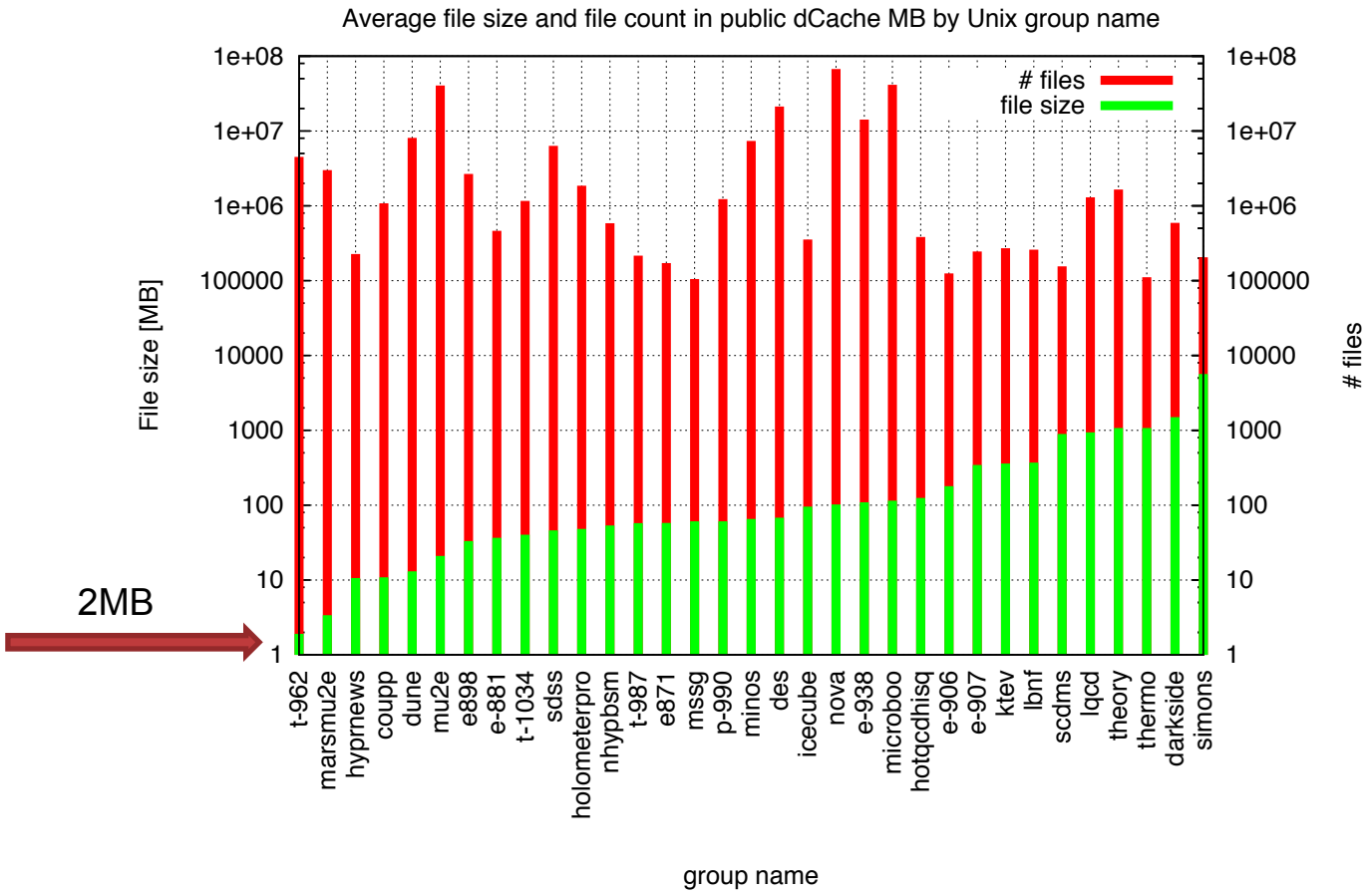


Entry type	count
file	232,010,959
dir	31,492,680
link	366759

Too many small files?

- In general not a huge problem, but
  - dump/restore takes long time
  - database schema upgrades take longer time
  - initial "df" command after NFS server start "hangs" noticeably
- We are approaching the issue of database size by trying to reduce each entry space overhead in dCache namespace DB. Forthcoming in 2.16 release.

Public dCache: File counts & avg. sizes



- All groups having > 100,000 files
- dCache is a distributed storage system and latencies associated with staging or even accessing each file over the network need to be taken into consideration. This is not a filesystem mounted off a local disk.

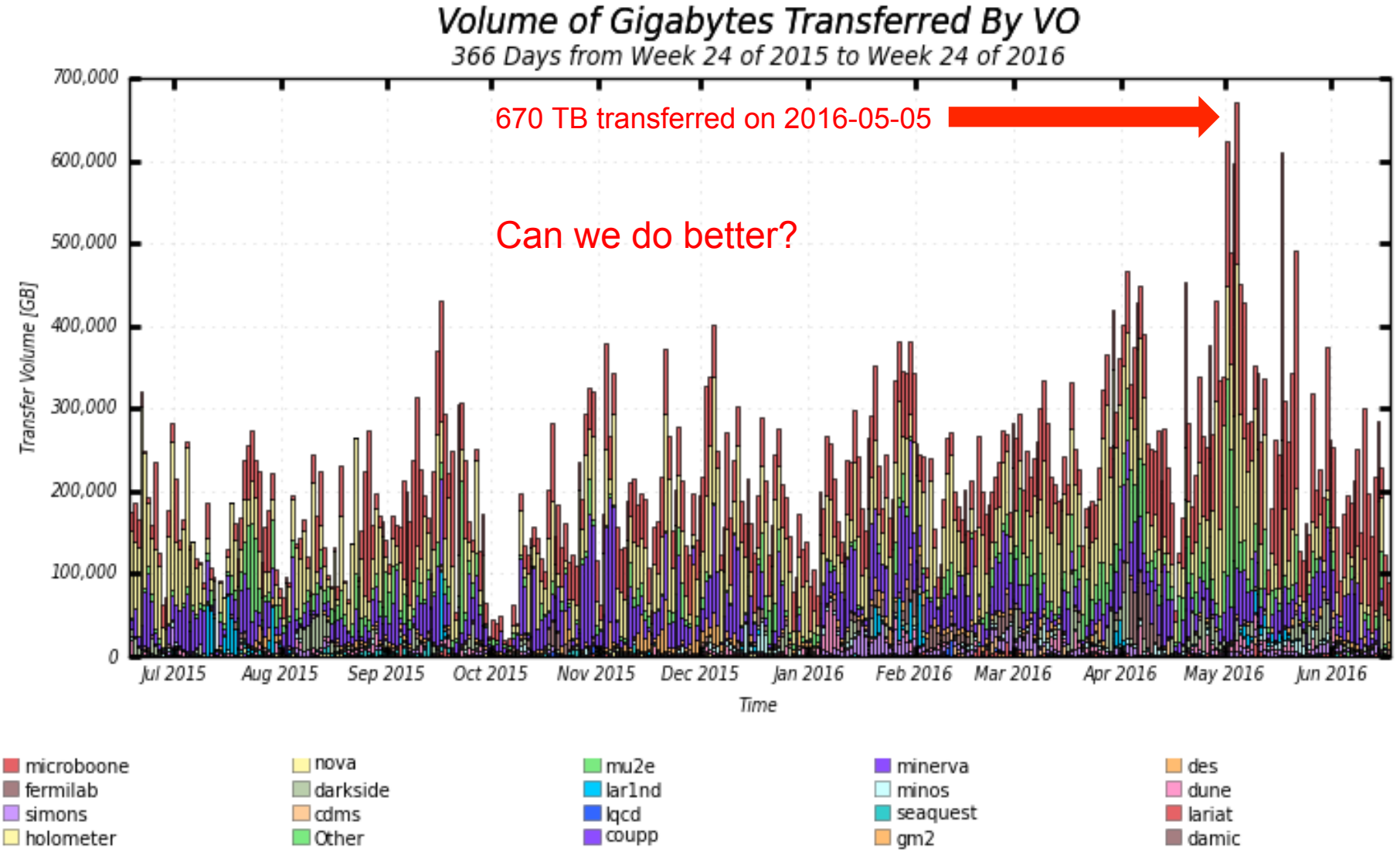
Best Practice: avoid millions of small files

- For the same amount of data small files cause:
  - Large number of files per directory.
  - Increase in depth and width of directory tree.
  - Datasets containing millions of files are harder to manage.
- When dealing with tape:
  - Per file overhead to write a file mark (on writes).
  - Tape back-hitching when data streaming to tape is interrupted (e.g. for the next file in queue) (on writes).
  - Mount latency (including load time) (on writes and reads).
  - Unload (rewind time) (on writes and reads).
  - Seek time (on writes and reads).
- dCache users are shielded from tape related overheads on writes as they occur out of band.
- Read tape overheads per file contribute directly to ''slow'' file delivery for files that are not cached on disk.
- Our current definition of small file is 200MB.
- Some small files may not belong to storage system (like file describing other file's metadata, or constants files etc.). These data best belong to a database (SAM or calibrations/conditions DB).

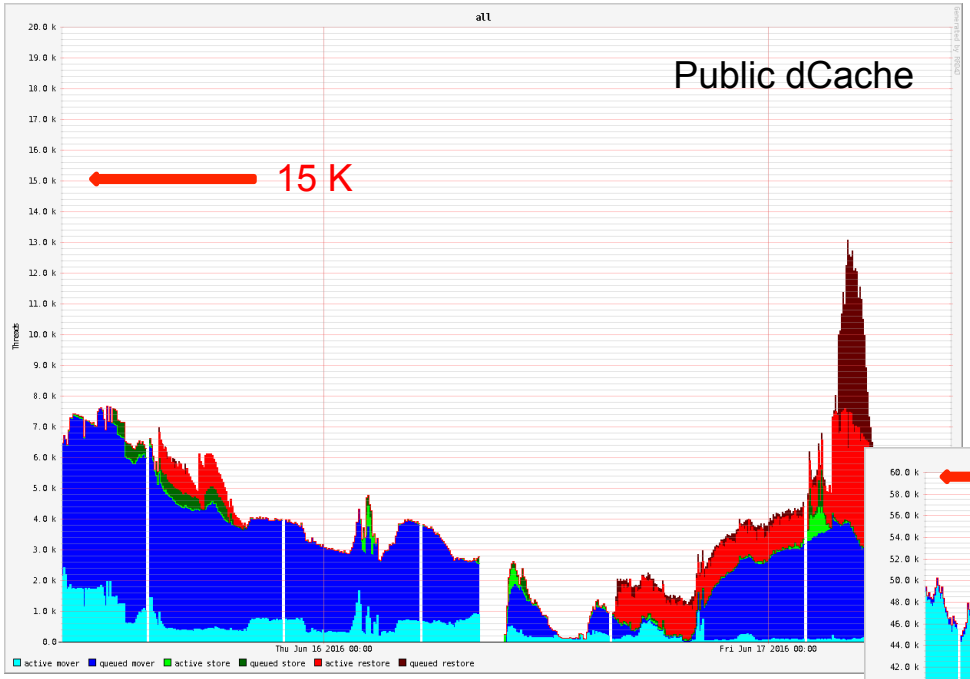
Small Files: pros and cons of SFA

- We offer Small File Aggregation Feature (SFA) in Enstore that automatically and seamlessly packs small files into large files that are stored to tape.
  - Pros:
    - User does not need to worry about small files.
    - Small files in the system have less impact on others when dealing with tapes as latencies associated with small files are minimized (less queueing in Enstore).
  - Cons:
    - Due to relatively random packing of files in SFA packages, **staging** small files back from tape may incur significant latencies when reading files if files are dependent (e.g. a data file and JSON file that describes data) and end up in different packages.
    - Writes to SFA are currently not distributed and performance is limited by a single appliance data server.
- Experiments are encouraged to consider packing or concatenating/merging their files intended for data analysis based on their own criteria rather than relying on automated SFA packaging:
  - Only experiments themselves know their data and can provide locality of reference (e.g. packaging aux. files together with their data file for efficient access).
- SFA is good for independent small files that do not require reading in specific sequence, so that in case they need to be staged there no dependency on files across packages.

Public dCache: peaks and valleys



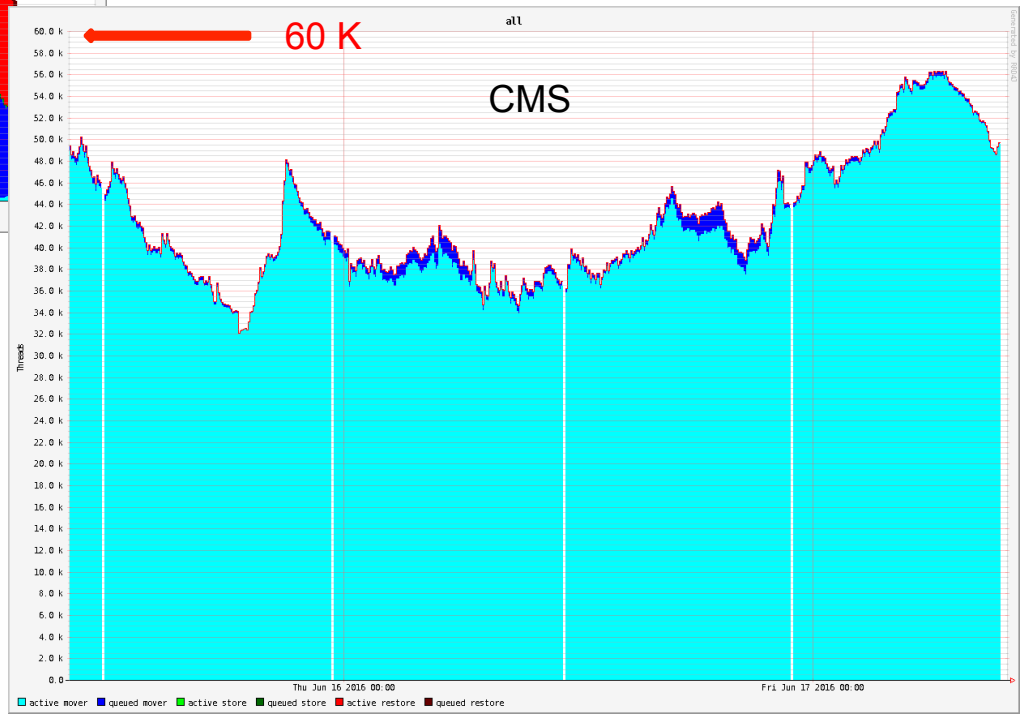
Public dCache vs CMS T1 disk



Public dCache : mostly queueing

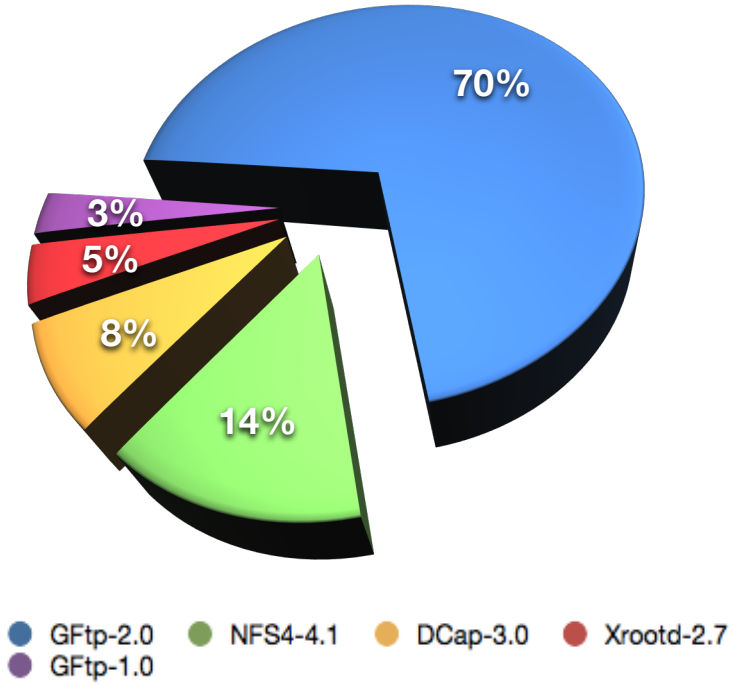
Why?

CMS : mostly running

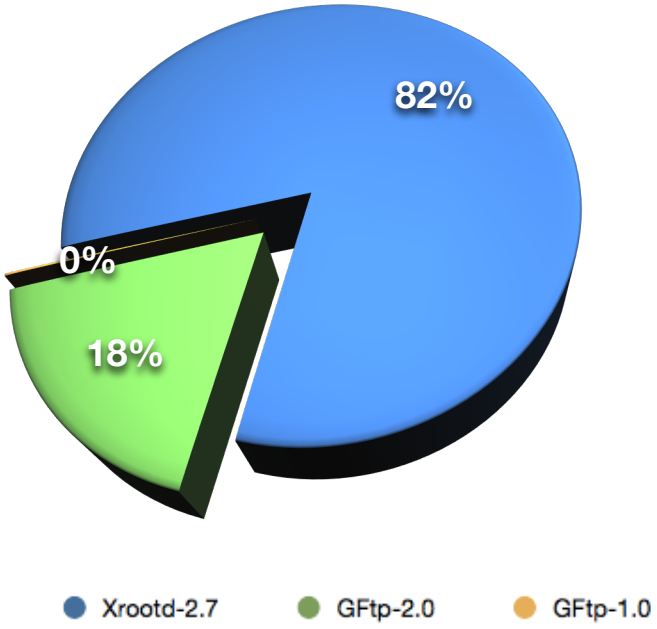


Protocols: breakdown of reads in the last 30 days

Public dCache

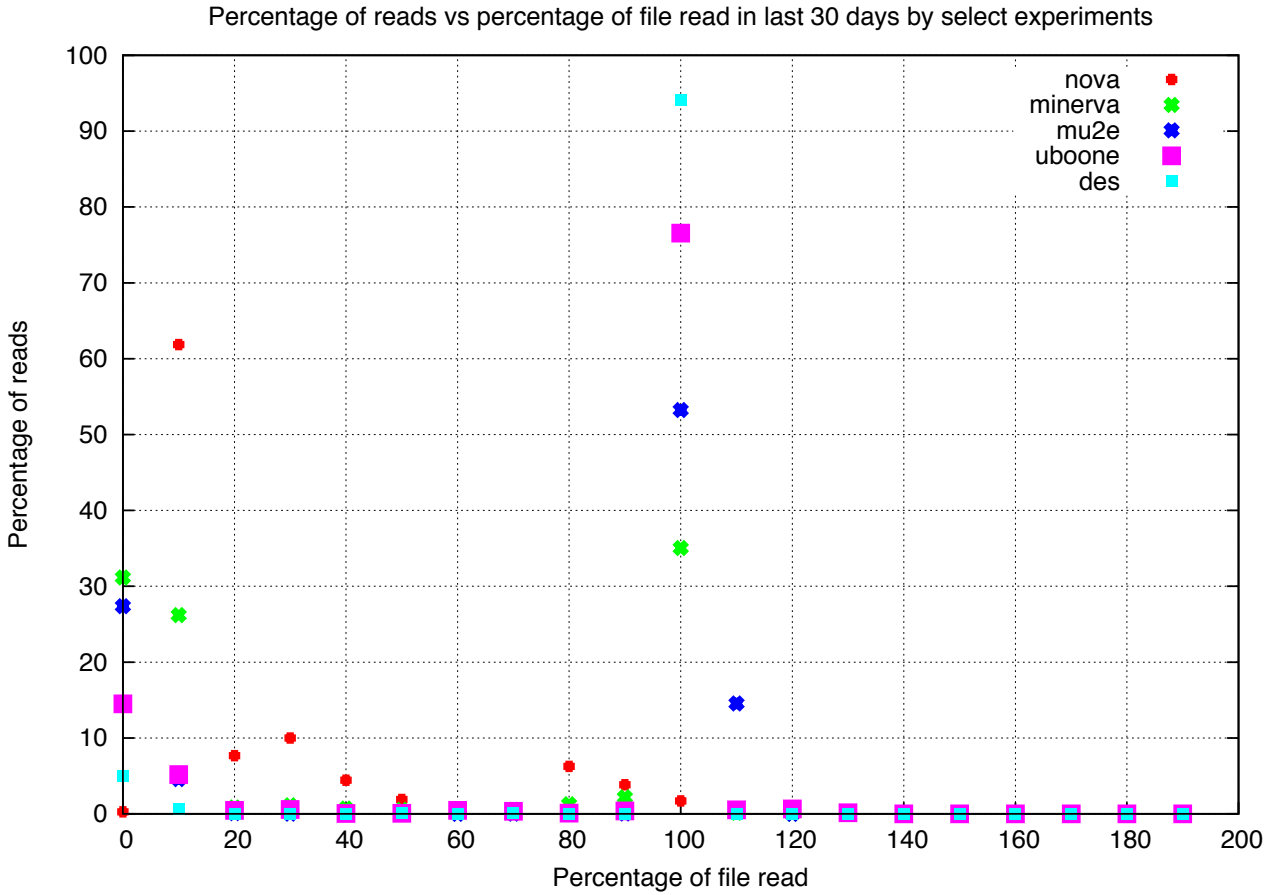


CMS T1 Disk



CMS is mostly streaming, Public dCache is mostly copying?

How much data is actually read



- Percentage of all reads vs percentage of file read by select experiments using “streaming” protocols like XRootD, NFS, DCap
- Copy would “win” if a file is read more than once (>100% in this plot)

Copy vs Streaming

- There would be no issue if pool nodes had infinite I/O and infinite bandwidth.
- We have to restrict number of active GFTP movers (transfers) per pool to 20 to avoid I/O subsystem overheating on pool nodes when they are hit by massive wave of fast transfers from CPU farms on Fermilab network or other fast networks.
- On the other hand slow transfers over the WAN occupy active slots for a long time eventually clogging up the system and leading to GFTP request queueing resulting in low job efficiency.
- Separating LAN and WAN traffic by directing them to different pools becomes cumbersome once there are many pool groups of different flavor belonging to different experiments.
- Streaming of data event by event does not stress pool I/O and allows opening of many more active mover slots per pool. E.g. we have 1000 XRootD active movers vs 20 GFTP per pool.

dCache best practice: do streaming (examples)

- E.g. instead of doing:

```
{cp,globus-url-copy,wget,xrdcp} /pnfs/foo/bar /local/disk
…
root[0] f = TFile::Open("/local/disk/bar");
```

- Do :

```
// access via xrootd
root[0] f = TFile::Open("root://fndca1/pnfs/fs/usr/foo/bar");
…
// or dcap
root[0] f = TFile::Open("dcap://fndca1/pnfs/fs/usr/foo/bar");
…
// or over NFS v4.1 or NFS v3 and dcap preload library
root[0] f = TFile::Open("/pnfs/fs/usr/foo/bar");
```

Best practice: run on ONLINE files to avoid queueing

- To keep job efficiency high and avoid wasting resources, and frustration it is best practice to run on ONLINE files, that is files that are in dCache pools.
- How to determine if file is ONLINE?

```
cat /pnfs/<experiment>/".(get)(<file name>)(locality)"
E.g:
cat ".(get)(ep047d08.0042dila)(locality)"
NEARLINE
```

- Meaning of file locality

file locality	meaning
ONLINE	file is only in disk cache (and not on tape)
ONLINE_AND_NEARLINE	file is in disk cache (and also on tape)
NEARLINE	file is not in cache (on tape)
UNAVAILABLE	file is unavailable (e.g. it is not on tape and the pool where it is located is down)

Public dCache: problems? we have a share of.

- 624 incident tickets assigned to “Storage Service” or “Data Movement development” since SNOW launched

dCache problems

- NFS issues pop-up from time to time though much less then pre 2.12. Often caused by unexpected behavior of NFS client, particularly when it comes to handling of local cache.
  - more fixes/workarounds on server side in 2.13
- Access to file is slow/hanging (using any protocol):
  - can be be caused by a number of things, but typically:
    - User is trying to access a file which is on tape and not in cache, expect latencies associated with tape access.
    - Source pool is not available (e.g. down) causing file to be fetched from tape; or transfer sleeps “indefinitely” (until pool becomes available) in case of non-tape backed files
    - Source pool is busy and maximum number of active transfers has been reached. In this case transfer is queued and will be scheduled on FIFO basis.
    - More rarely caused by a failure of SPOF dCache component or external component(s) (like GUMS server)

NFS issues, best practices

- A serious one, reported on incident INC000000681473. Silent data corruption when using NFS v4.1. When pool is badly overloaded we have seen cases when copies in dCache are corrupted with no error generated. Correct size, wrong checksum. Still under investigation. A handful of cases.
- Best practice is to compare Alder32 checksum of original file and its copy in PNFS after you copied file in dCache.
- Only SRM protocol does it automatically.
- How to extract checksum of a file from PNFS?

```
cat /pnfs/<experiment>/".(get)(<file name>)(checksum)"
E.g.:
cat /pnfs/fs/usr/test/litvinse/world_readable/".(get)(ep047d08.0042dila)(checksum)"
ADLER32:fe99e83e
```

A word on NFS

- dCache NFS v4.1 interface provides a convenient POSIX-like way to access your data. This convenience helps to acquire certain habits that may interfere with scalability and portability of experiment's data handling framework as it evolves and data volume grows:
  - Reliance on mounted NFS v4.1 limits analysis to Fermilab domain.
  - Building file catalogs based on directory trees soon runs into scalability issues. Use SAM for dataset management.
  - Data access model that is deeply based on POSIX access to files will have issues with adopting new protocols/technologies that could be more performant than pNFS access (less of an issue if data access is based on root with its pluggable I/O interface).
- Realizing these pitfalls and taking into account evolving industry trend of moving away from POSIX the SCSA committee came up with Fermilab's Computing Storage Architecture for HTC (presented yesterday by Gerard). The recommendations:
  - Do not support POSIX access on worker nodes.
  - Retain POSIX access only for interactive analysis or in case by case basis.

Best Practice: read the manual

- We are trying to keep current the “How to” document:
  - **Using Public dCache at Fermilab**
    http://cd-docdb.fnal.gov/cgi-bin/ShowDocument?docid=5399

Summary

- Avoid creating millions of small (under 200MB files).
  - Consider all effects of SFA before deicing to rely on it fully for data processing.
  - Merge/concatenate/package your files based on your workflows.
- Streaming vs copying.
  - Prefer streaming.
- End-to-end checksumming.
- Use ONLINE files.

Extra Slides

Dealing with client side timeouts

- If your transfer are often queued or files happen to be on tape only in many cases client-side timeout limits get tripped. How to handle this?
- Here are two examples :
  - GFTP, use:

```
-stall-timeout | -st <seconds>
   How long before canceling/restarting a
   transfer with no data
   movement.  Set to 0 to disable.
   Default is 600 seconds.
```

  - root access to files via xrootd, add this line to your .rootrc file:

```
XNet.MaxRedirectCount: 255
```

  - if using xrdcp add

```
xrdcp ... –DIRedirectLimit 255 -DIRequestTimeout 14400 ...
```

NFS issues, best practices

- "Unknown Error 523" or, Input/Output error when listing directories.

```
ls /pnfs/exper/foo
ls: reading directory /pnfs/exper/foo: Unknown error 523
```

- Or Input/Output error (depending on SLF version):

```
ls /pnfs/exper/foo
ls: reading directory /pnfs/exper/foo: Input/output error
```

- Remedy is to cause kernel to drop clean caches:

```
echo 3 > /proc/sys/vm/drop_caches
```

(requires root access though)

NFS issues, best practices

- If NFS is extremely slow and there is high load. Client host is otherwise healthy. Any access to NFS ends up in ‘D’ state.
  - Open SNOW ticket specifying your host name. We should be able to reset this mount on NFS server side w/o you having to resort to rebooting your host.