dCache Deployment What works for USCMS

Jon Bakken
March 30, 2009

dCache Cells

I've been asked to report on our dCache deployment here at Fermilab. Setting up an optimized dCache is far from obvious, & I'm happy to share our layout. Your dCache layout may be different.

- Rule 1: There are admin cells & IO cells - never mix them.
  - The most common mistake is putting gridftp doors on admin nodes
  - IO cells (and nodes) can go up & down, but when admin cells go up & down, the effects can ripple throughout the entire system
- Important Admin cells - separate node for each:
  - "Home" admin cells - PoolManager, LocationManager, LoginBroker,
    Broadcast Cell & general PAM cell
  - PNFS server & PNFSManager should be on the same node
  - SRM + srm components + PinManager should be on the same node
  - I typically run multiple dCap doors on a node, & generally don't mix other other cells with dCap cells. dCap is very scalable & I allow 4000 sessions/door.
- Other Admin cells - need to be on different nodes than main cells, but can be combined as needed: gPlazma, 2 replica managers, Info Provider/Collector, Httpd, Billing, etc.
- Up-to-date dCache batch files available at http://cmsdcam2.fnal.gov/dcache/batch/batchlist.html
- We are in the process of replacing all admin nodes with new machines - before run starts
- Planning on moving all admin nodes to 64-bit SL5 next Tuesday. (Pnfs requires code changes)
- Detailed layout on next page.

Admin Current Physical Deployment

Node	Bought	Type	GB	Use
cmsdca0	Jul05	SL4,i686	4	PoolManager, AdminDoor, LocationManager, Broadcast, PAM, LoginBroker (hsmcontrol when it works)
cmsdca	Jul07	SL4,x86_64	8	4 R/O-dCap, 3 Restricted-dCap
cmsdca1	Jul07	SL4,x86_64	8	4 R/O-dCap, 4 Restricted-dCap
cmsdca3	Aug08	SL4,x86_64	16	3 R/O-dCap, 3 Restricted-dCap
cmsdca2	Jul07	SL4,x86_64	8	SRM, PinManager, ThreadManager, GSIFtpManager, CopyManager, GSI-PAM
cmsdca4	Jul07	SL4,x86_64	8	gPlazma, 2 ReplicaManagers, postgres database for ReplicaManagers
cmspnfs1	Jan08	SL4,x86_64	32	PNFS Server, PnfsManager, dir, Cleaner Postgres DB for PNFS + Companion
cmspnfs3	Jun07	SL4,x86_64	12	PNFS Hot space + database backup
cmssrv57	Jul07	SL4,x86_64	8	Postgres database for PNFS+PinManager
cmsdcam	Jul07	SL4,x86_64	8	Httpd, Billing, Statistics, Topo, WebCollector, PoolCollector, InfoCollector, InfoProvider Postgres db for billing
cmsdcam2	Jul05	SL4,x86_64	4	General Montoring Node
cmsdcam3	Jul07	SL4,x86_64	8	SRMWatch, General Monitoring, SE-GIP

PNFS Server

We replace our PNFS server every 18-24 months - very critical node.
We typically spend ~20K$ on this single node to ensure adequate performance.
Pay very close attention to independent disk layout for databases & logs & system.

We spent more than 2 years working the obscure bugs out of PNFS when we deployed dCache/Enstore for RunII (CDF/D0). We reached a point where we had stable operations -- we are very very leery of moving to Chimera at this point in the LHC program.

- Moving to Chimera also requires changes in Enstore databases, a lab-wide change.
- Decided FNAL won't be the among the 1st to move to Chimera - not even planned right now
- We have tools in place to track down "bad" users - those doing finds, recursive ls, etc.

We found that we can get dramatic PNFS server performance increases if we:

- Set all logging parameters in pnfsSetup to 0 (no logging)
- Redirect all log output from the PNFS Server to /dev/null
  - These last 2 items made us nervous (no logs!), but we've found that this made PNFS reliable & we didn't have to check for errors any more.
- We also compile the PNFS server on the node we are going to run it on
- We had to disable client authentication - we found tremendous & unexplainable amounts of context switching during PNFS server calls. Patrick changed PNFS & added authentication as an option. Disabling authentication reduced context switching significantly. ----> "FastPnfs" option, available at dCache.org web site

We are planning on switching from Postgres to BerkeleyDB as underlying pnfs database on April 7.

- Factor of 4 speed boost

Databases

Putting databases on the node local to the accessing cell generally provides the highest throughput.

- Especially true for PNFS, & PNFSManager
- Only exception for us is the SRM/Pin database - we run this on a separate node

All of our databases are password protected - we use pgpass - this means the password are not in the batch or setup files. (Then they are in a specified root-owned 400 permission file)

Do not write logs & database data files to the same partition - big hit in performance

Raid level of database disk is important too - documentation says Raid5 is bad. Raid10 is good. This has a big effect again on performance.

Lots of database details, including parameter settings we use in talk “Postgres Basics” - I presented this talk at the dCache workshop recently in Karlsruhe. I attached it to this meeting’s talks as well.

SRM Deployment

Based on advice from Timur, we do not run the standard deployment of the SRM

- (Have not understood why this was never propagated back to general use)

Run SRM + PinManager + all cells in the utility JVM (the RemoteTransferManagers & CopyManagers) inside the Tomcat web application

- Timur's reason was based on evidence that the communication for messages passed within a single JVM are much faster than those passed between JVMs.
  - Since SRM talks to PinManager & TransferManagers, this led to a large performance boost.
- We've been running this way for ~3 years & it has worked well for us

Also based on Timur's advice, we run the database for SRM/Pinning on a separate node.

- Only case where we run a database not on the 'local' node.
- Needed because of the large resource consumption by the SRM

Note: We do not run SpaceManager at all - the concepts of space tokens don't exist.

SRM Parameters

Because of the large number of transfers, we need to increase the default SRM parameters. Here are the list of our changes:

acceptCount=10000 in /opt/d-cache/libexec/apache-tomcat-5.5.20/conf/server.xml, BIG recent change

```
gsiftpMaxStreamsPerClient=20
srmBufferSize=2097152
srmTcpBufferSize=2097152
remoteGsiftpIoQueue=WAN
remoteGsiftpMaxTransfers=2000
srmCopyReqThreadPoolSize=2000
performanceMarkerPeriod=30

gsidcapIoQueue=default
srmDbLogEnabled=true
pnfsSrmPath=/pnfs/fnal.gov/usr/cms/WAX
useGPlazmaAuthorizationModule=true
useGPlazmaAuthorizationCell=true
srmProxiesDirectory='${homeRoot}'/dcache-proxies
srmVacuum=false
```

There are of course corresponding changes in sysctl.conf for tuning the kernel TCP parameters.

Use dcache.local.run.sh & dcache.local.sh to customize the startup of services

dcache.local.run.sh:

- check for correct deployment of tape services on node
- check if pnfs mounted
- check if pool structure correct
- check if pool writable
- check that certs (CA+CRL) valid
- check for obscure case of data loss (files cached & control file in mixed state)

dcache.local.sh:

- set ulimit 65K

We run with 4GB JVM for SRM, & 4 GB JVM for Httpd, 1.5 GB JVM for dCache, & 256K for lm, otherwise we use the standard 512 MB JVM for all other cells

Twice an hour we query the dCache for its status, & we retry old (many hours) transfers

- Internally retry "No-Mover-Found"
- Internally retry Stuck P2P transfers
- Internally retry "Waiting" movers

We kill all transfers older than 2.5 days. (Batch job limit is 2 days)

Pool Current Physical Deployment

~3 PB of disk (Nexsan SataBeasts) 656 different pools on 129 nodes

- We configure max of 11 TB/pool - balance with startup time (we do not use Berkeley DB)
- We configure pools to use all but 100 GB of the unix partition space.
- Newer nodes have 3 pools/node, Older nodes have 8 pools/nodes,
- All new disk configured with RAID-6, older disks converted slowly as needed
- Every pool node has a 2 GB bonded network connection, all on public network

- Pools divided into primary IO section, unmerged resilient section, user small-file resilient section, & stage-in from tape section. For us, segregating the stage-in to a separate section was a huge performance boost. All pools can stage-out to tape.

- Movers: 2 main queues - LAN & WAN
  - Before lazy download: 1800-LAN, 50-WAN, 150-P2P movers per pool
  - After lazy download: 25-LAN, 10-WAN, 50-P2P

One GridFTPDoors run on all pool nodes except pools staging files in from tape

We do CRC transfer checking based on cksum value stored in TMDB, & ignore dCache crcs for WAN

- Provided similar tool for users doing local transfer as well

Make sure you have updatedb.conf configured so locate doesn't index files on pool disk or pnfs

Mover Queues

Rule #2 - Your system won't work robustly unless your data is equally divided amongst all your pools.

- We run space equalizers to move data between pools - goal is same amount of space used in each pool.
- Then, with equally used pools as a given, we've found that a random selection of a data pool is the best choice. This continues to distribute data equally & provides the highest bandwidth.
- If data is written uniformly across the pools, reads are also of course optimized
- Key is to use all the hardware you purchased at once.

Before Lazy download, it was possible to be rather cavalier & set a very high number of movers in the LAN queue - primarily because the Posix IO on the workers only moved a few bytes at a time.

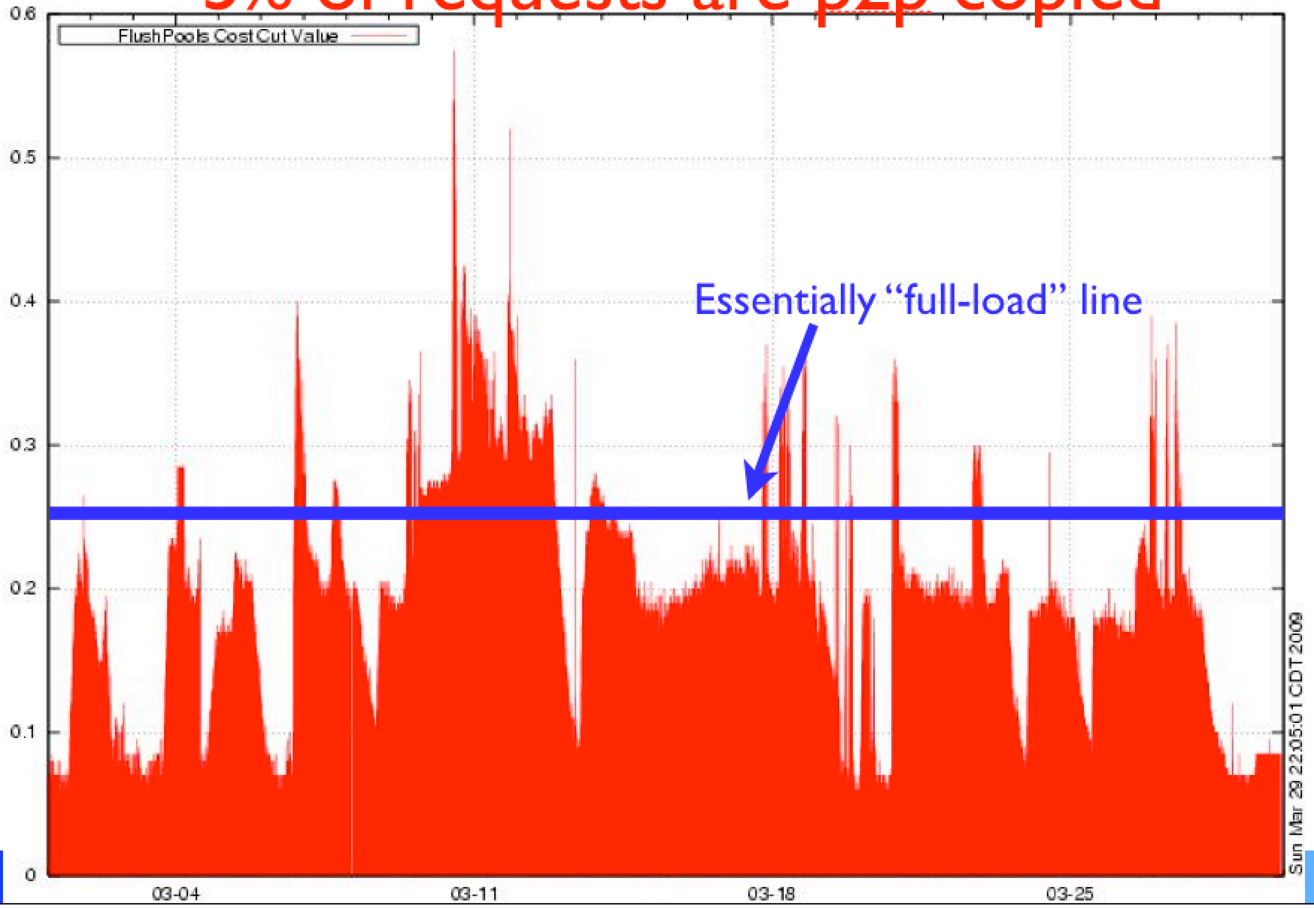
With Lazy download, large portions of the file are transferred at once & one needs to limit the active movers on the LAN queue or the node will crash due to resource exhaustion.

- Unfortunately, the lazy download mode keeps a mover slot open until it exits, sometimes a very long time later. This can lead to mover queuing.
- Wrote a script that runs every 5 minutes that checks for queuing & tries increasing & quickly set back to nominal the max allowed mover value.
  - Script acts like a dam's spillway & tries to prevent overflows(queuing) & floods (resource exhaustion)
  - By appropriate tuning of timeout parameters, we can start new movers on pools that lazy downloads have finished & not on ones that still have active transfers.
  - Naive idea works wonderfully. Works only if majority of LAN transfers are Lazy Download

Cost Cut

dCache only provides a static cost cut value - primarily used to determine whether to queue request or do a pool-to-pool copy.

- A static value is simply insufficient to properly tune varying user load.
- Wrote a simple script that dynamically tunes cost cut every 5 minutes so at most 5% of all transfers are pool-to-pool copied. Cut down on "instantaneous" mover queues significantly.
- dCache Cost Cut value so only 5% of requests are p2p copied



Essentially "full-load" line