Update on Storage.

Handling of Scientific Data at DESY, Location Zeuthen

Stephan Wiesand
DESY – DV –

Technical Seminar
Zeuthen, 2010-06-29

HELMHOLTZ GEMEINSCHAFT

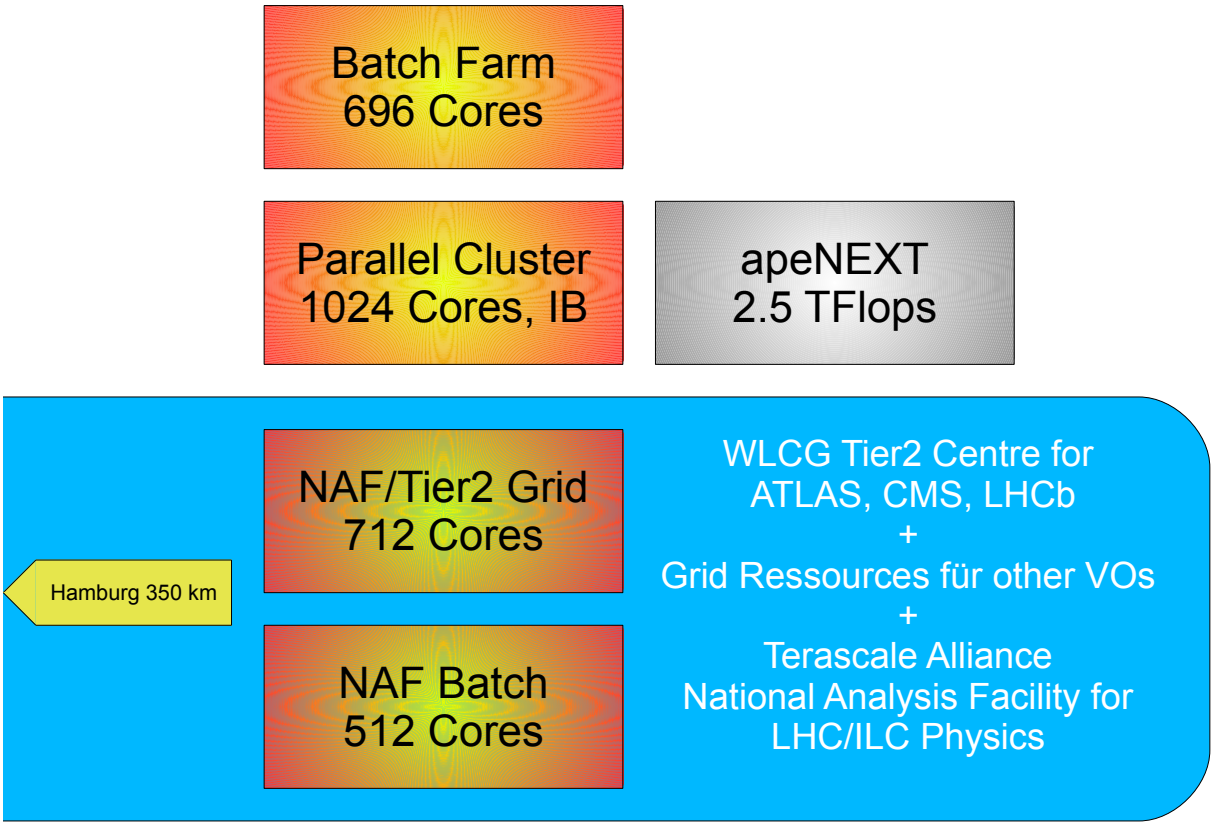
DESY

Agenda

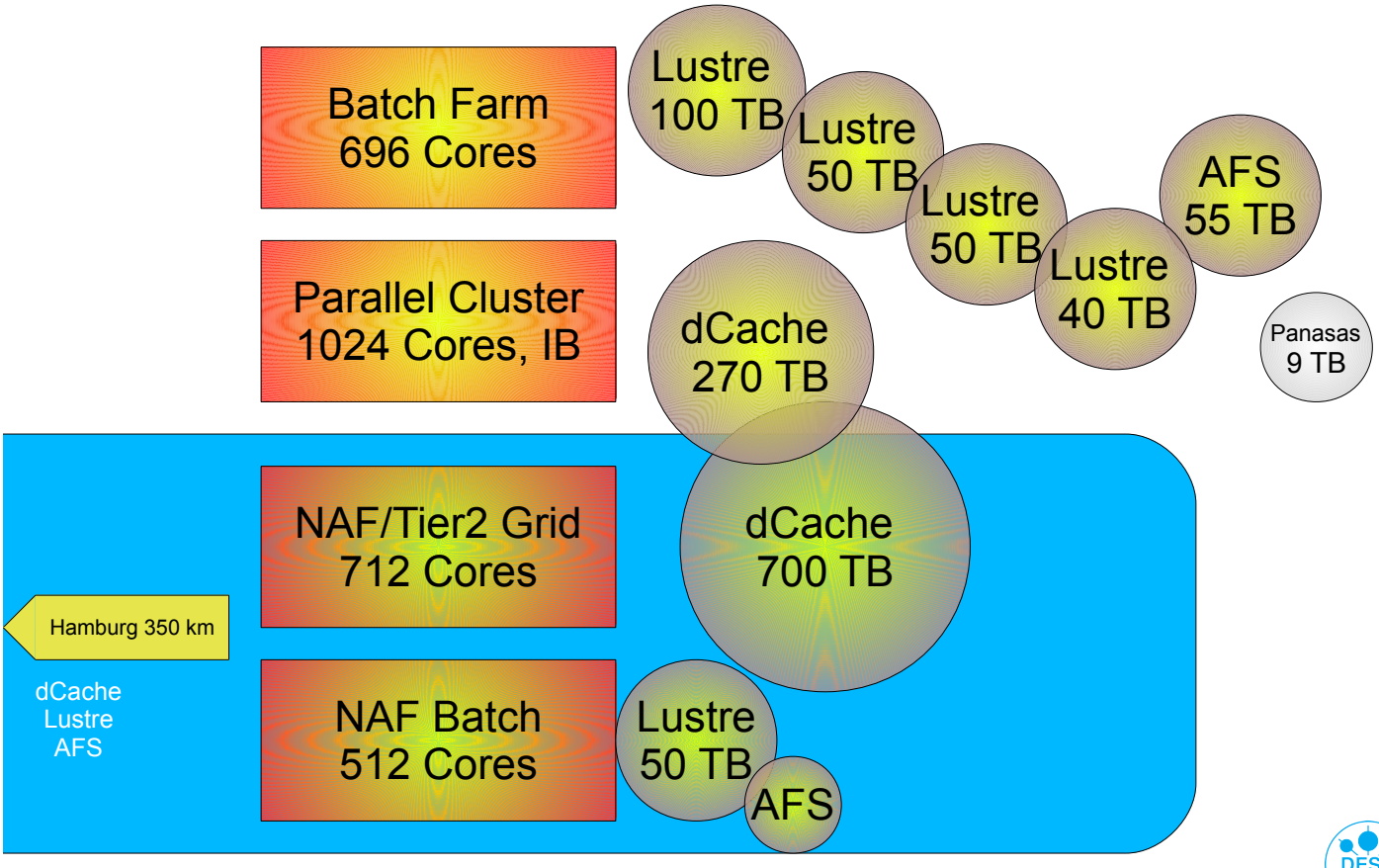
- topic: bulk data
  - event data, simulation results, lattice configurations
  - derived datasets (ntuples) , calibration data, ...
- technology of storage solutions used in Zeuthen
  - filesystems
    - AFS, Lustre, dCache
  - hardware
- implications for
  - efficient use
  - planning
- alternative & future solutions

DESY

Computing at DESY, Location Zeuthen



Computing + Disk Storage at DESY, Location Zeuthen



The Storage Brick

- direct Attached Storage. Typical configuration:

- OS: S5L 64-bit
  - automatic, central installation, configuration, maintenance, monitoring
  - just as for compute nodes (all systems fully patched)

Another Typical Configuration

- server + 12 x 3.5” disks in a 2 HU box

- 20 TB raw net capacity at RAID6 with 2 TB SATA drives
  - up to 0.4 PB in a single rack (peak power consumption ~ 8 kW)
- or: 1.4 TB raw net capacity with 146 GB 15k SAS drives
  - ~ 20 x performance/capacity for streaming access
    - even better for random I/O
- several configurations in between
- => can tailor hardware configuration to application needs
  - general tradeoff: speed vs. cost/space/power

Advantages of Direct Attached Storage

- compared to large storage devices behind a SAN:
  - cost
    - x 2 … x 10
  - performance
  - simplicity
    - leveraging existing know how & methods, including monitoring
    - as already used for compute nodes & other servers
  - incremental growth
    - at current
      - market price
      - performance
      - space density
      - power efficiency
    - hardware configuration tailored to actual use case
    - rapid purchase and deployment

Data and Metadata

- data: the actual file content
- metadata: information about a file
  - filename, parent directory (=> path)
  - ownership
  - permissions
  - location
- AFS, Lustre, dCache allow aggregating file servers
  - into a single namespace
- common concept to do this: separating data and metadata
- => typical: data scales very well, metadata doesn't
  - but different filesystems behave differently
  - notice data : metadata ~ average file size

AFS

Volume Location Database cluster at application level

Fileservers

- volume based
  - namespace is constructed from embedded mount points
  - R/O replication, asynchronous
  - transparent migration
  - volume quotas (2 TB max)
- metadata:
  - volume location data: small amount, low transaction rate
    - no scalability problems (at our size)
  - per file metadata resides on the fileserver, within the volume
    - scales ok

AFS: Advantages

- reasonably secure
- available on farm, cluster, WGS, PC
- group space administration delegated to group admins
  - afs_admin
- backup selectable per volume (matching quota)
  - separate group quotas for space with/without backup
  - files from backup can be retrieved by users
- easy to separate user groups/activities (dedicated fileservers)
- usable ACLs (per directory), working the same way on each client
- clients available for Linux, Windows & others (OS X, Solaris)
- metadata transaction capacity scales with number of fileservers

AFS: Disadvantages

- AFS token required for authenticated access
  - expires
- client relatively slow
  - persistent client side cache helps in some cases, hurts in others
  - has much improved in recent years, more improvements soon
    - we do not recommend to use Atrans/afscp any more
      - will be removed from our systems soon
- volumes are confined to their fileserver partition
  - data is not distributed over fileservers automatically
  - not file by file (or even stripe by stripe)
  - scalable throughput can still be achieved
    - but requires distribution of data over volumes
      - and smart placement of those on different servers
        - does not work in practice

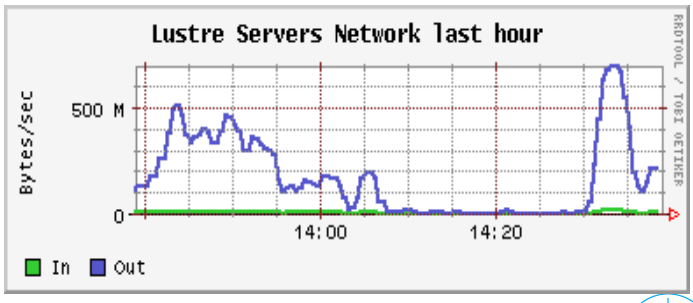
Lustre

Metadata Server

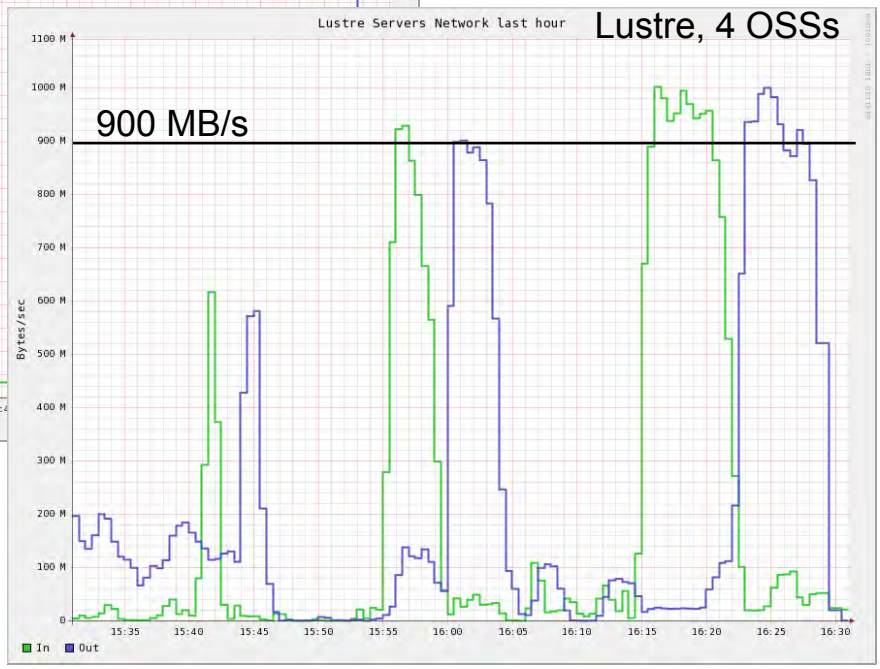
- looks like a single POSIX filesystem to the client
- files are distributed round robin across OSTs when created
  - automatically
- single files can even be striped across OSTs (not advisable for common usage)

Object Storage Servers

- real life performance of our first Lustre instance:
  (3 OSSs with 2 x 1 GbE each)

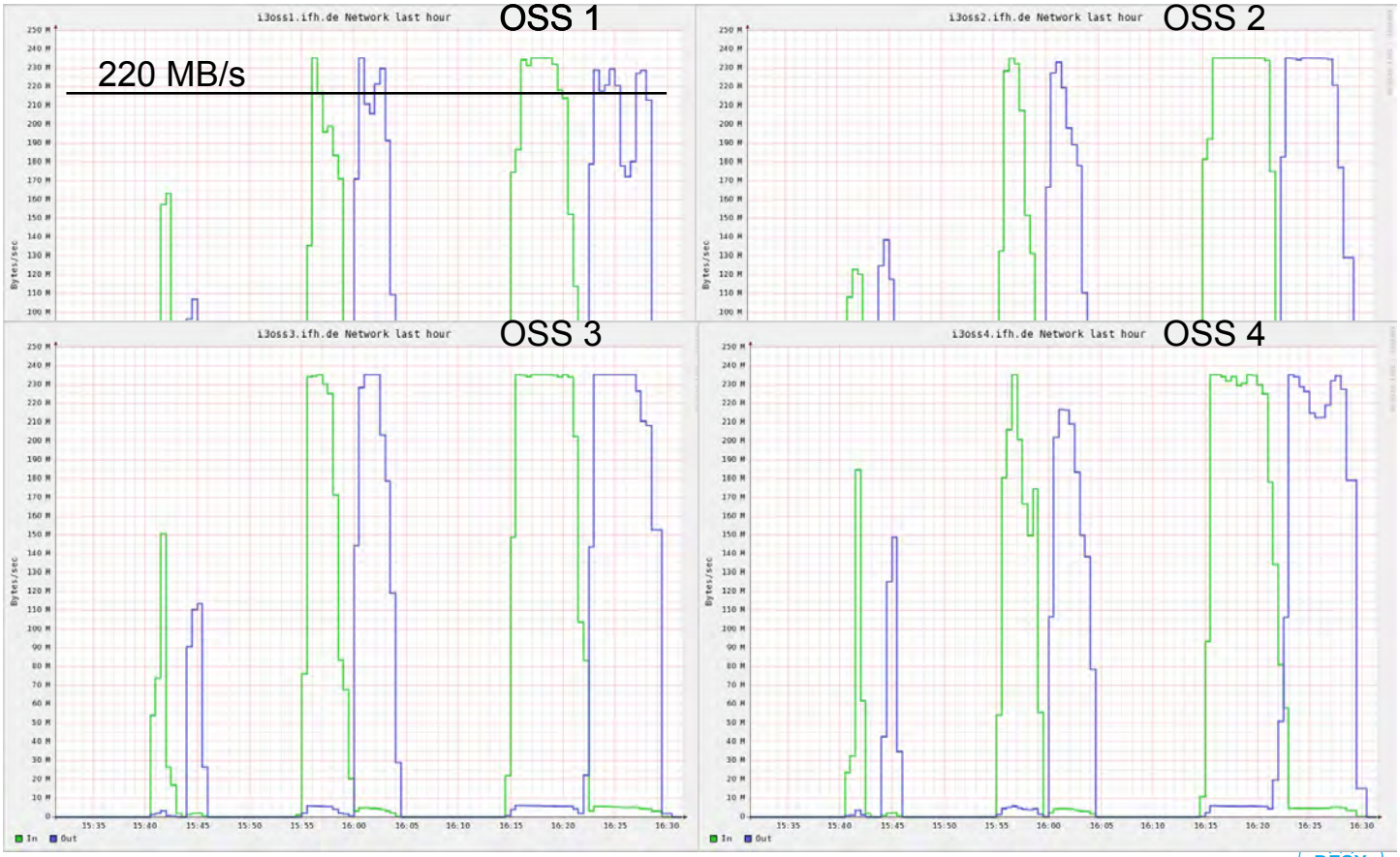


Performance: AFS vs. Lustre in Burn-In Tests



- 64 Clients, 128 Jobs
- copy/read 2 GB each

Lustre Burn-In Test



Lustre: Advantages

- high & scalable data performance, large filesystems
  - without hassle for users
- fast client
  - single client easily saturates a GbE connection
  - uses the operating system cache
- supports modern, fast interconnects
  - in particular: Infiniband
    - have seen 500 MB/s for a single client-server connection
- multihomed servers & clients possible
  - fast infiniband access from some clients to some servers
  - ordinary ethernet for other combinations
- more useful features on the roadmap

Lustre: Disadvantages

- public roadmap no longer exists
  - future slightly unclear
- not as mature as other filesystems yet
  - does not cope well with network problems
- missing features
  - transparent migration, replication
  - security (anything better than auth_sys)
    - can only be made available to trusted clients over trusted networks
      - farm, cluster, WGS - not PCs, notebooks, foreign clients
- ACLs: POSIX draft not as useful as AFS ACLs, and harder to use
- quota: user/group quota not as useful as volume concept
- tight coupling of servers and clients
  - client crash significant event causing delays for other clients

Lustre: “Problem”

- metadata for each and every file resides on a single MDS
  - aggregate lookup/open/create performance limited by single server
  - can be a real problem if many clients rapidly access different files
- a small file (say,1 kB) takes up as much space on the MDS as on the OSS
  - and accessing it probably causes more work on the MDS
- => not suitable for (many) small files
- storing large amounts of data in small files is always a bad idea
  - but on Lustre, it's particularly bad
    - performance can easily become worse than with AFS
- storing a TB in 100 byte chunks should not be done using files
  - use a database instead

dCache

- not an “ordinary” filesystem
  - files can not be modified

Head Node

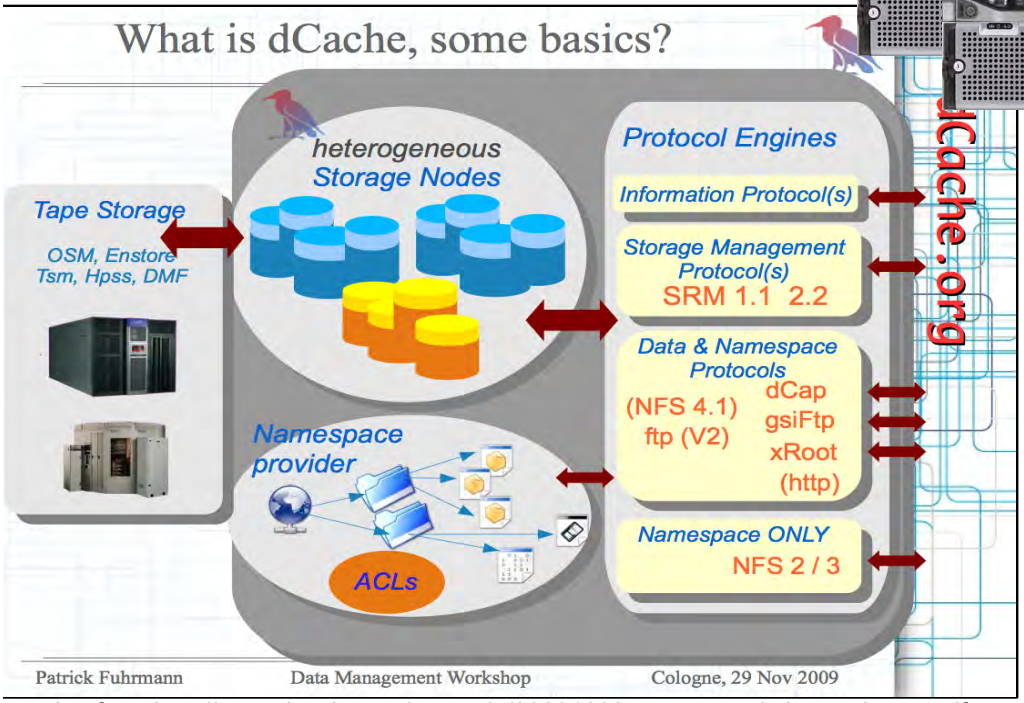
Pool Nodes

dCache

- not an “ordinary” filesystem
  - files can not be modified

Head Node

Pool Nodes



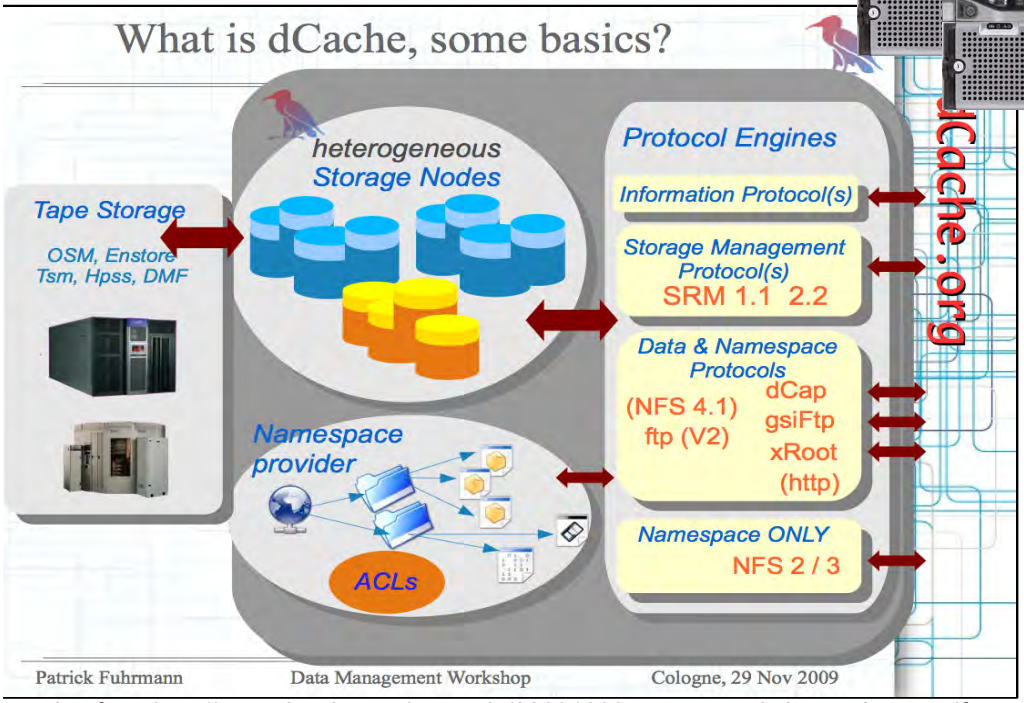
stolen from http://www.dcache.org/manuals/20091030-storageworkshop-cologne.pdf

dCache

- not an “ordinary” filesystem
  - files can not be modified

Head Node

Pool Nodes

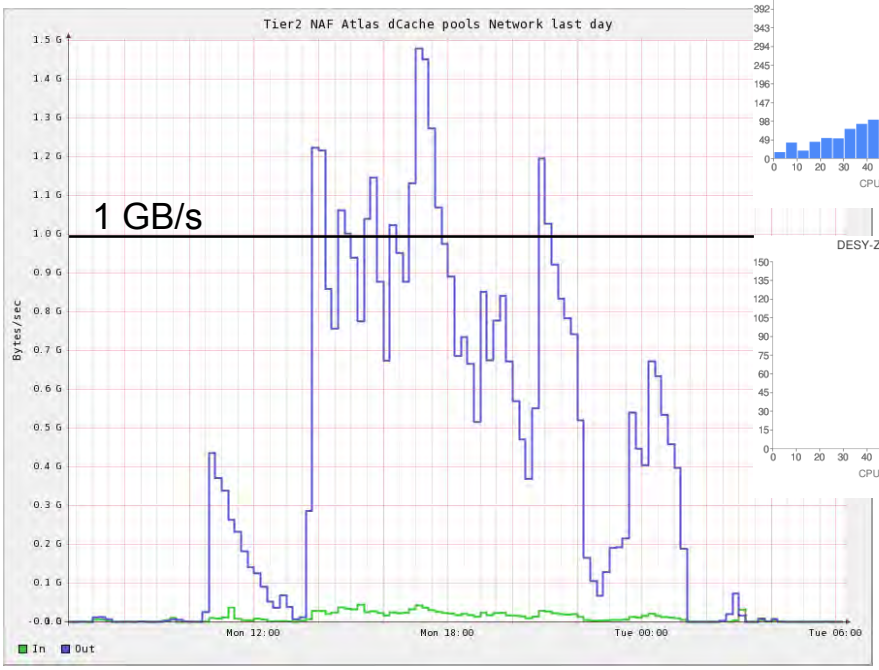


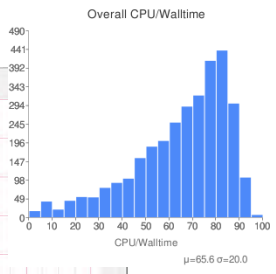
stolen from http://www.dcache.org/manuals/20091030-storageworkshop-cologne.pdf

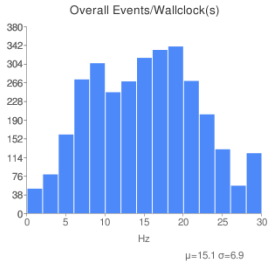
- replication
- migration
- HSM optional
- R/O pools
- grid storage element (SRM)

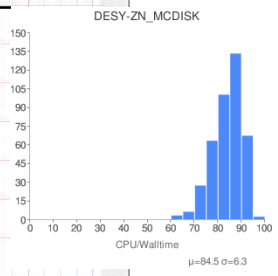
ATLAS Hammercloud Tier2 Site Test, March 2nd, 2010

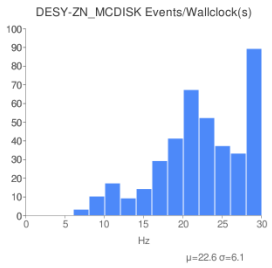
> dCache throughput











http://gangarobot.cern.ch/hc/1131/test/

dCache Throughput Test

dCache: Setup Options

- classic: disk cache in front of tape storage
  - dedicated read & write pools
    - cheap read pools, best quality write pools
  - or general purpose pools
  - disk space is reused according to “least recently used” policy
    - but pinning files is possible
  - files no longer available in a read pool can be “prestaged”
    - contact uco if planned for large number of files (efficiency)
- can just as well be used without tape backend
- pools are dedicated to storage groups (one or more)
- files can be cloned automatically
  - to 2nd tape, for precious data
  - to other disk pools, to improve resilience and/or performance

dCache: Access

- no access with the normal tools or libraries like cp, open(), …
- pnfs
  - nfsv2 export by head node mounted on /acs on our clients
  - provides POSIX-like access to the namespace only
    - ls works, but cp still doesn't
- native access: dcap (dCache access protocol)
  - dc_open(), dc_read(), … calls from libdcap
  - some HEP applications (like ROOT) come with dcap support
- the preload library libpdcap enables access with dynamically linked, normal applications
  - does not work well with all applications
  - deprecated, library no longer maintained

Example: Accessing Files in dCache

- copy to local disk

```
% dccp /acs/users/wiesand/Event.root /tmp
```

- using ROOTs native dcap support:

```
% root
[…]
root [0] f=TFile::Open("dcache:///acs/users/wiesand/Event.root")
```

- using the preload library:

```
% export LD_PRELOAD=/opt/products/dcache/default/lib64/libpdcap.so
% root
[…]
root [0] f=TFile::Open("/acs/users/wiesand/Event.root")
```

- may look similar
  - but very different under the hood
  - prefer native access if possible

Grid Access to dCache

- get a transfer URL for the desired protocol, then use it:
  - dcap

```
% lcg-gt srm://lcg-se0.ifh.de/pnfs/ifh.de/data/atlas/users/ahaupt/data.1m dcap
dcap://lcg-dc0.ifh.de:22125//pnfs/ifh.de/data/atlas/users/ahaupt/data.1m

dccp dcap://lcg-dc0.ifh.de:22125/pnfs/ifh.de/data/atlas/users/ahaupt/data.1m /tmp/test
1048576 bytes in 0 seconds
```

  - gsidcap

```
% lcg-gt srm://lcg-se0.ifh.de/pnfs/ifh.de/data/atlas/users/ahaupt/data.1m gsidcap
gsidcap://lcg-se0.ifh.de:22128//pnfs/ifh.de/data/atlas/users/ahaupt/data.1m

% dccp gsidcap://lcg-se0.ifh.de:22128/pnfs/ifh.de/data/atlas/users/ahaupt/data.1m /tmp/test
1048576 bytes in 0 seconds
```

  - gsiftp

```
% lcg-gt srm://lcg-se0.ifh.de/pnfs/ifh.de/data/atlas/users/ahaupt/data.1m gsiftp
gsiftp://ssu36.ifh.de:2811//pnfs/ifh.de/data/atlas/users/ahaupt/data.1m

% globus-url-copy gsiftp://ssu36.ifh.de:2811//pnfs/ifh.de/data/atlas/users/ahaupt/data.1m \
file:///tmp/test
```

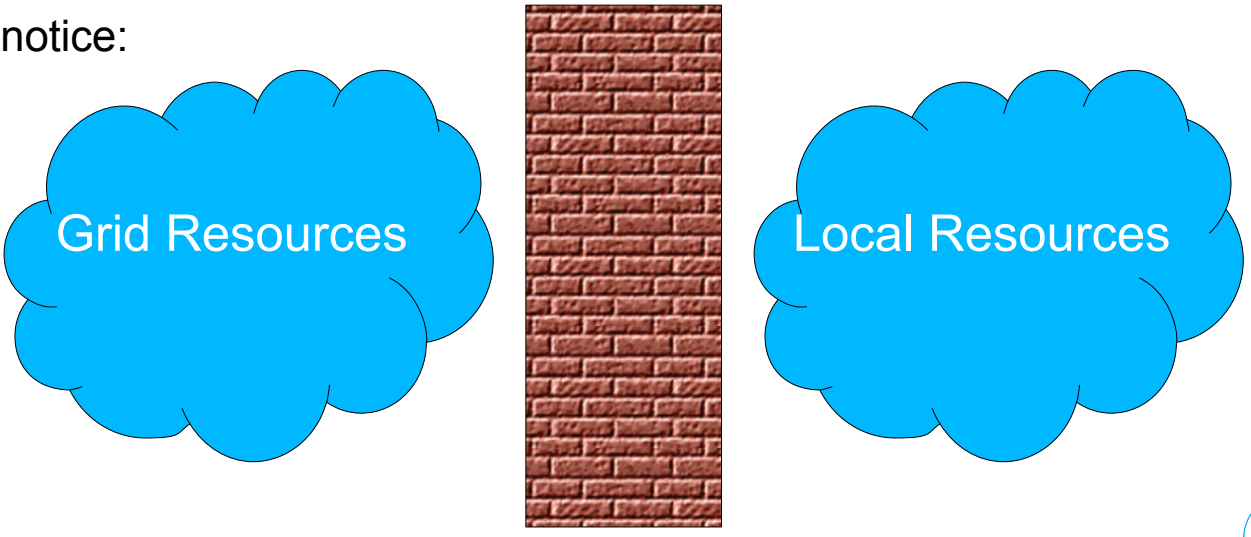
  - srm

```
% lcg-cp srm://lcg-se0.ifh.de/pnfs/ifh.de/data/atlas/users/ahaupt/data.1m file:///tmp/test

% srmcp -streams_num=1 srm://lcg-se0.ifh.de:8443/pnfs/ifh.de/data/atlas/users/ahaupt/data.1m
file:////tmp/test
```

dCache / SRM: Beware of Firewalls

- commands on last slide are available after **`ini glite`**
- important to access files from firewalled clients:
  - **`export DCACHE_CLIENT_ACTIVE=1`**
    - by default, the pool node tries to connect to the client
    - for the same reason, `srmcp` requires **`-stream_nums=1`** to work
- notice:



dCache: Advantages

- most versatile
- many different access options
  - local access via dcap, gsidcap
    - pnfs available on central systems only (farm, cluster, WGS)
  - access from anywhere via gsiftp, srm
    - all our dCache storage is grid-enabled
  - in future, will add WebDAV, pNFS (NFS 4.1)
- very good aggregate performance

dCache: Disadvantages

- no immediate POSIX access
  - pNFS will remedy this, but may take a while
- files cannot be modified, only deleted and rewritten
  - this won't change
- modest single client performance, no Infiniband support
- Head Node is equivalent to Lustre MDS
  - single point of failure
  - limits scalability
  - dCache is no more suitable for small files than Lustre
    - especially with tape backend
      - small files do not belong on tape
        - abysmal performance
        - wear & tear due to shoe shining, mount operations

Alternatives & Possible Future Options

- free:
  - PVFS (open source, from Argonne & Clemson Universisty)
    - simple parallel filesystem deliberately sacrificing features
      - no locks
  - FHGFS (closed source, binary only, available for RHEL)
    - parallel filesystem from Fraunhofer Society
    - commercial support available for a fee
- commercial:
  - Panasas
  - GPFS, optional tape backend with HPSS (IBM)
  - supported Lustre storage from Oracle (, HP, DDN)
- under development:
  - AFS/OSD

Recall: AFS

Volume Location Database cluster at application level

- volume based
  - namespace is constructed from embedded mount points
  - R/O replication, asynchronous
  - transparent migration
  - volume quotas (2 TB max)

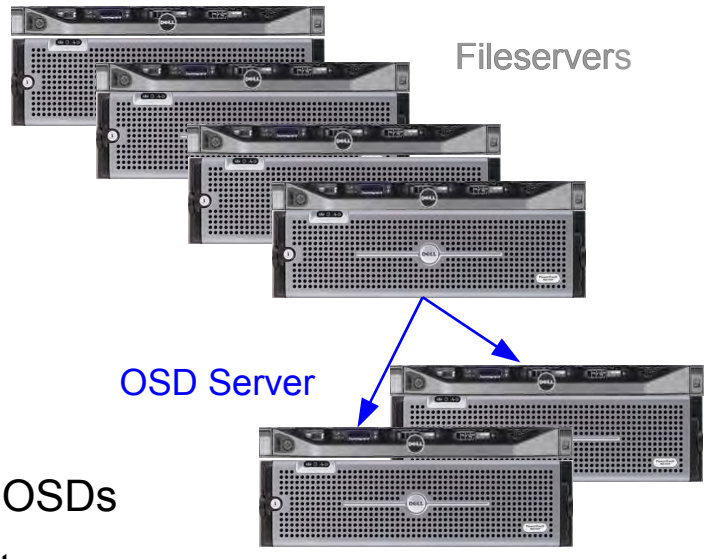
Fileservers

- metadata:
  - volume location data: small amount, low transaction rate
    - no scalability problems (at our size)
  - per file metadata resides on the fileserver, within the volume
    - scales ok

AFS + OSD - A Promising Development

Volume Location Database cluster at application level

- volume based
  - namespace is constructed from embedded mount points
  - R/O replication, asynchronous
  - transparent migration
  - volume quotas (2 TB max)



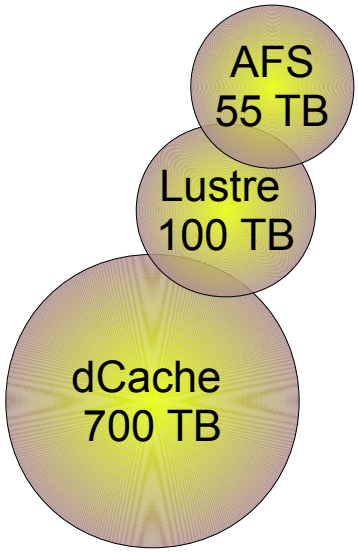
- small files stored on fileserver
- large files stored (or striped) on OSDs
- parallel access to OSDs by clients
  - possibly with direct access to backing filesystem (Lustre, GPFS)
- http://www.rzg.mpg.de/projects/hsm-afs

Conclusion

- AFS, Lustre, dCache all have their strengths and weaknesses
  - probably true for **any** filesystem, including commercial solutions
  - no silver bullet
- but a viable solution is available for all use cases
  - except for tons of small files
- current options: 3 filesystems (x) many hardware configurations
- the key to success is finding the right setup for a project
- best practice for new deployments:
  - meeting of a few project members with -DV- storage experts
    - to find out the actual requirements
    - and the most suitable solution

Summary

- From common storage bricks using DAS, flexible storage solutions are built to users' needs.
- This Ansatz and the three Filesystems are doing well in practice.
- Solutions based on Lustre and dCache can be very performant.
- AFS is not going to break any speed records. It has other virtues though. And with the OSD enhancement, it could become a very good compromise for many use cases.
- The most important ingredient is communication between users and providers of storage.



Final Remark: About Using Desktop PCs for Storing Data

- single SATA drive
  - PC class
    - not meant for heavy duty
- no redundant power
- no UPS
- no backup
- possibly physical access by others
- very limited monitoring
- no consistency checks
- not accessible except locally
  - ssh possible - except when someone else turned off the PC
- => just don't do it