Why XROOTD ?

(An Overview of Storage Element)

Pavel Jakl[1]

[1] Nuclear Physics Institute, Academy of Sciences of the Czech Republic

STAR Collaboration meeting

MIT
Boston, USA

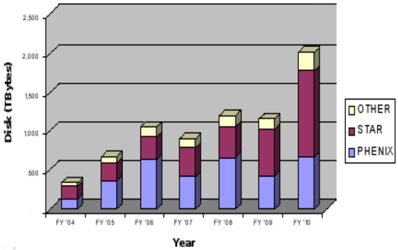
14th of July 2006

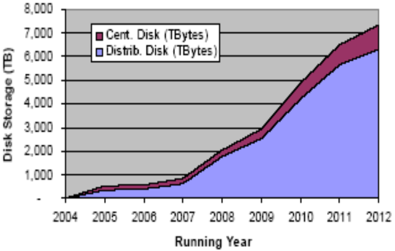
Outline

1. Hardware vs Software approach
2. Xrootd and recent work
3. Ongoing/future work
4. Summary

Distributed vs centralized storage

- over 1PB data per year (in plot just raw data)
- present-day resources:
  - centralized disk space(NFS area): **75 TB**
  - distributed disk space(spread on 320 nodes): **130 TB**
- **distributed** vs centralized disk:
  - ⊕ very low cost (factor of $\sim$10)
  - ⊕ less human resources to maintain
  - ⊖ worse manageability (sometimes called "Islands of Information")
  - ⊖ none of current data management solutions allow to directly exploit distributed storage





Native NFS solution for distributed disk ?

- thousands of random concurrent accesses from end users batch jobs would overcome scalability of NFS architecture
- NFS infrastructure nor software does offer efficient load balancing among data servers
- lack of any fault-tolerance for missing or corrupted data in native NFS
- no solution scheme for finding other copies of missing file (sometimes called LFN to PFN resolution)
- when any of NFS servers has troubles, remote machines will experience problems
- several hardware improved NFS-like solutions exist:
  - ◇ Panasas file system (PanFS)
  - ◇ General Parallel File System (GPFS)
  - ◇ Lustre

PanFS, GPFS, Lustre (Hardware approach)

	PanFS	**GPFS**	**Lustre**
Scalability (clients)	no limit	up to 4096	no limit
Load balancing	Storage Blades	Distributed meta-data	Distributed lock manager
Performance(IO read/write)	fast	slowest	fastest
Supported protocols	NFS, CIFS, *DirectFlow*^{TM}	NFS, CIFS	NFS, CIFS
Network infrastructure	Ethernet	Ethernet, Fibre Channel	Ethernet, Quadrics
Linux kernel version	2.4	2.4, 2.6	2.4, 2.6
OS Linux dependency	RHEL	RHEL, SUSE	RHEL, SUSE
Tape migration	-	HPSS	-

Is it rootd efficient ?

1. ROOTD knows only PFN
   - rootd doesn't know where the data is located -> data needs to be cataloged and kept up-to-date
2. Overloaded and not responding node
   - rootd connection will expire after defined time and job will die
3. Job start time latency
   - catalog is not updated accordingly when node is down for maintenance
   - job dies when requested files are deleted between the time "a" job is submitted and starts
4. Static data population
   - human interaction is needed to populate data from HPSS to distributed area
   - datasets need to be watched (datasets get "smaller" in case of data loss)
5. Write access and authorization issue
   - everyone in rootd is "trusted" user (missing authorization)

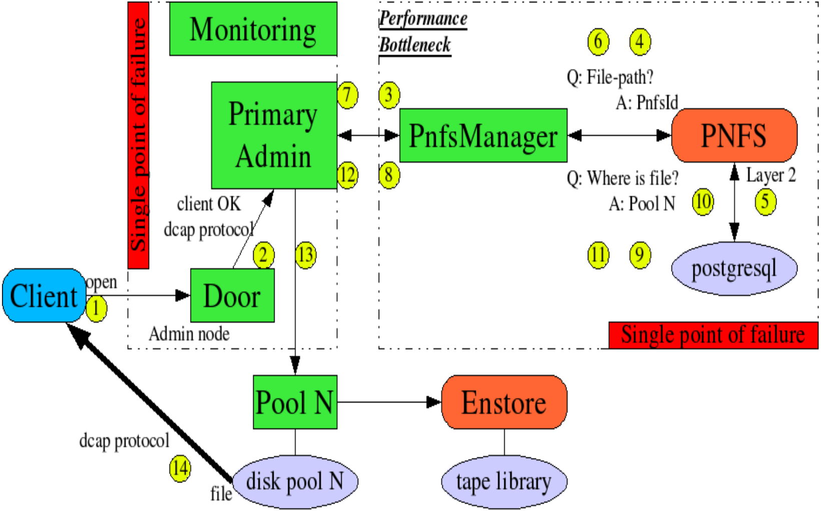
What is Xrootd and Dcache ?

- distributed file systems providing high performance file-based access
- main goals:
  - Scalability - can serve thousands of clients
  - Fault-tolerant - adaptation to server crash or missing data
  - Flexible security - allowing to run any security protocol
  - Load balancing - sharing the load among multiple servers
  - MSS integration - accessing files from permanent storage (such as HPSS)
  - Single global unique name-space - span single name-space across multiple servers
  - Replica management - determination of the location and multiplicity of data
  - Grid integration - possibility to talk to other data management tools

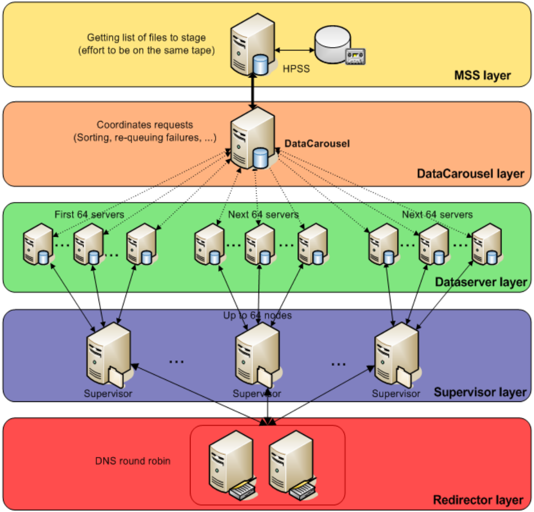
ROOTD, XROOTD, DCache (Software approach)

	ROOTD	DCACHE	XROOTD
Developed by	ROOT	DESY & FNAL	SLAC, BNL, INFN
Scalability (nodes)	no limits	no limits	no limits
Security	uid/gid	Kerberos	any authentication
Platforms	all platforms	all platforms	all platforms
Fault-tolerance	No	MSS plugin	MSS plugin
Replica management	No	Yes	Yes
MSS plugin	No	Yes	Yes
Authorization	No	Yes	Yes
Load balancing	No	No	Yes
Protocol	No	dCap	xroot
Grid integration	No	SRM (frontend)	SRM (frontend)
Single point of failure	almost each node	Namespace handling, head node	No ?

DCache Overview



Xrootd Overview

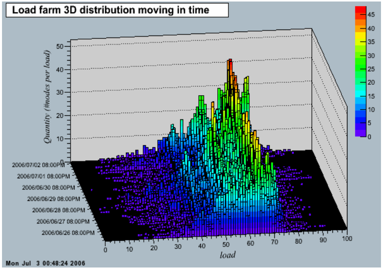


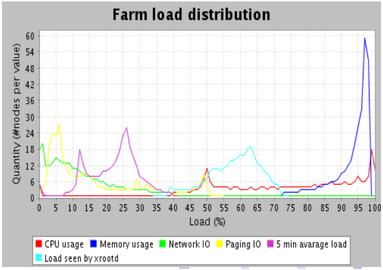
Issues review

- needed to teach xrootd LFN and rootd PFN handling for backward compatibility with rootd
- un-coordinated requests from xrootd led to HPSS downtime
    - resolved by integrating DataCarousel into XROOTD
- need for having automatic purging in absence of free space
    - setting and testing thresholds + partial re-implementation
- HPSS IO Rate was too slow (3MB/s) -> server was selected several times at the same time
    - needed to understand load balancing for server selection in case of file restore from HPSS

Understanding load balancing

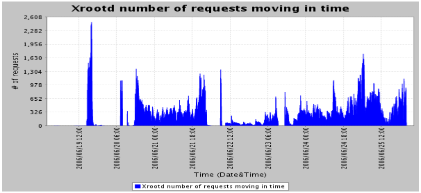
- what should be considered as a load ?
  - CPU, memory, network usage or load caused by system ?
- computation of an overall load is flexible:
  - it is a combination of 5 main information (cpu,memory etc.) reported by each node
- relative weight and combination of those 5 parameter have been chosen by implemented plot visualization reflecting our enviroment

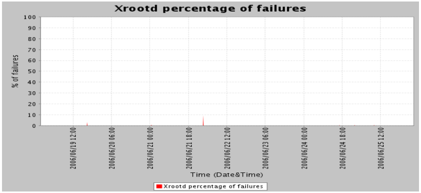




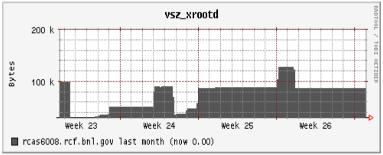
Monitoring XROOTD

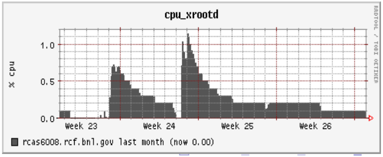
- xrootd seems now very stable and scalable in terms of serving data access requests



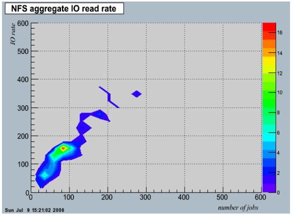


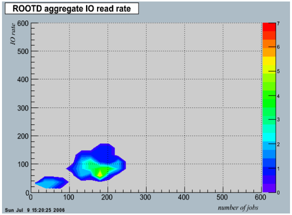
- the monitoring of XROOTD behavior in large scale and over long period of time haven't shown significant impact on CPU or memory consumption on nodes

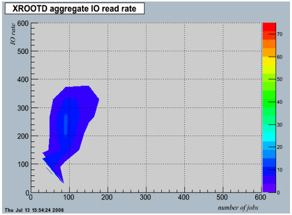




Preliminary aggregate IO Comparison







Motivation . . .

XROOTD is not perfect and could be extended:

- does not bring files over from other space management systems (dCache, Castor etc.)
- always bring files from MSS, not from neighboring cache
- in large scale pools of nodes, clients could ALL ask for a file restore: lack of coordination or request "queue"
- no advanced reservation of space, no extended policies per users or role based
- no guarantee for stored files (no lifetime, no pinning of files)
- only access files (what about event-based access ?)
- other middleware are designed for space management. Leveraging on other projects and targeted re-usable components ?

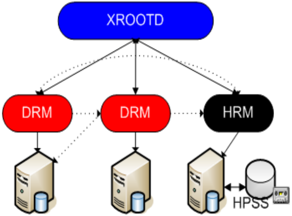
SRM funcionality

- **SRM:** the grid middleware component whose function is to provide dynamic space allocation and file management on shared distributed storage systems
  - Manage **space**
    - Negotiate and assign space to users and manage *lifetime* of spaces
  - Manage **files** on behalf of user
    - Pin files in storage till they are released
    - Manage *lifetime* of files
  - Manage file sharing
    - Policies on what should reside on a storage or what to evict
  - Bring the files from remote locations
  - Manage multi-file requests
    - a brokering function: queue file requests, pre-stage

XROOTD+SRM integration plan

Types of storage resource managers:

- Disk Resource Manager (DRM)
  - Manages one or more disk resources
- Tape Resource Manager (TRM)
  - Manages the tertiary storage system (e.g. HPSS)
- Hierarchical Resource Manager (HRM=TRM+HRM)
  - An SRM that stages files from tertiary storage into its disk cache



- resting on SRM development, plan to add an SRM layer behind XROOTD
- xrootd is responsible for managing the disk cluster
- DRM is responsible for managing the disk cache
- HRM is responsible for staging files from MSS

Summary

- Xrootd is deployed on 320 nodes (the biggest production deployment of xrootd)
- load balancing and handshake with HPSS make the system resilient to failures
- few months of running xrootd in production mode helped to stabilize the current version
  - thanks to F. Simon, A. Kocoloski and M. van Leuween for feedback
- next version of scheduler will allow access files from HPSS
- preliminary measurement of aggregate IO shows better results than rootd and even Panasas
- un-coordinated requests to MSS were interim resolved by DataCarousel (future HRM backend)
- integration with SRM should be done at the end of September 2006

Questions ?