Data access on widely distributed worker nodes

(using Xrootd/Scalla and SRM)

Pavel Jakl[1,2] Jérôme Lauret[1] Andrew Hanushevsky[4]
Arie Shoshani[5] Alex Sim[5] Alexandru Romosan[5]

[1]Brookhaven National Laboratory, United states of America

[2]Nuclear Physics Institute, Academy of Science, Czech Republic

[4]Stanford Linear Accelerator Center, United states of America

[5]Lawrence Berkeley National Laboratory, United states of America

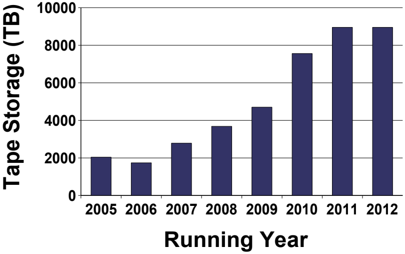
for STAR collaboration

ROOT workshop 2007
CERN, Geneva

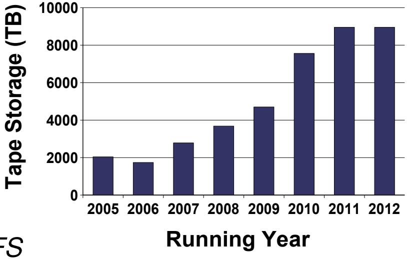
Outline

- *Storage challenges at STAR experiment*
- *Past years experience and data model*
- *Xrootd real production scenario and performance tuning*
- *XROOTD+SRM integration*
- *Summary*

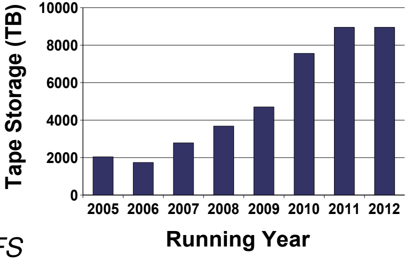
- over 1PB data per year at STAR



- over 1PB data per year at STAR
- *Permanent* location:
  - tape system (HPSS): offers several **PBs**
- *Temporary* locations:
  - centralized disk space: **75 TB** *via NFS*
  - distributed disk space: **350 TB** *spread over 500 nodes*

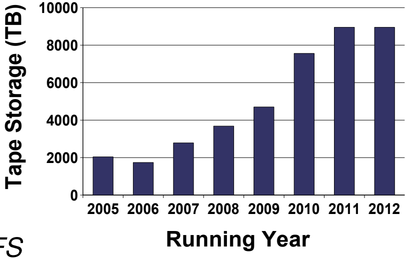


- over 1PB data per year at STAR
- *Permanent* location:
  - tape system (HPSS): offers several **PBs**
- *Temporary* locations:
  - centralized disk space: **75 TB** *via NFS*
  - distributed disk space: **350 TB** *spread over 500 nodes*

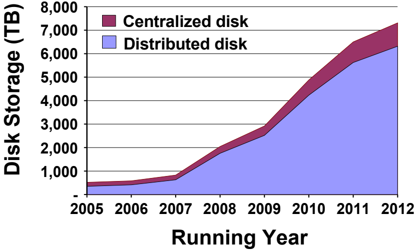


- **distributed** vs centralized disk:
  - ⊕ very low cost (factor of ∼10)
  - ⊕ less human resources to maintain
  - ⊖ worse manageability (one has to build aggregation)
  - ⊖ no native OS/system provides scalable/workable solution

- over 1PB data per year at STAR
- *Permanent* location:
  - tape system (HPSS): offers several **PBs**
- *Temporary* locations:
  - centralized disk space: **75 TB** *via NFS*
  - distributed disk space: **350 TB** *spread over 500 nodes*

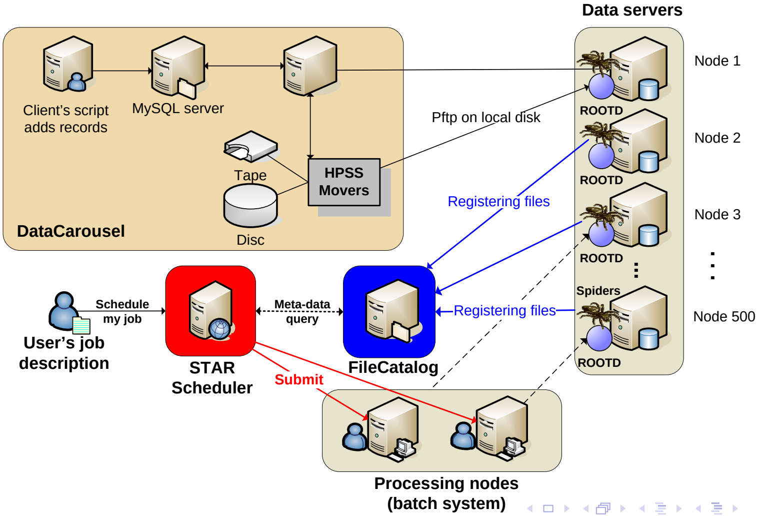


- **distributed** vs centralized disk:
  - ⊕ very low cost (factor of ∼10)
  - ⊕ less human resources to maintain
  - ⊖ worse manageability (one has to build aggregation)
  - ⊖ no native OS/system provides scalable/workable solution



ROOTD based (old) data model in STAR

ROOTD - provides remote file access mechanism via data server daemon



Is it rootd scalable and maintainable ?

1. ROOTD knows only PFN (**P**hysical **F**ile **N**ame)
   - rootd doesn't know where the data are located -> data needs to be cataloged and kept up-to-date
2. "Spidering" scalability issues
   - additional problems with "Spidering" of nodes when the storage grows (too many processes, db deadlocks etc.)
3. Overloaded and not responding node
   - rootd connection expires after defined time and job dies
4. 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
5. Static data population
   - human interaction is needed to populate data from HPSS to distributed area
   - data-sets need to be watch (data-set gets "smaller" in case of disk reset/format)

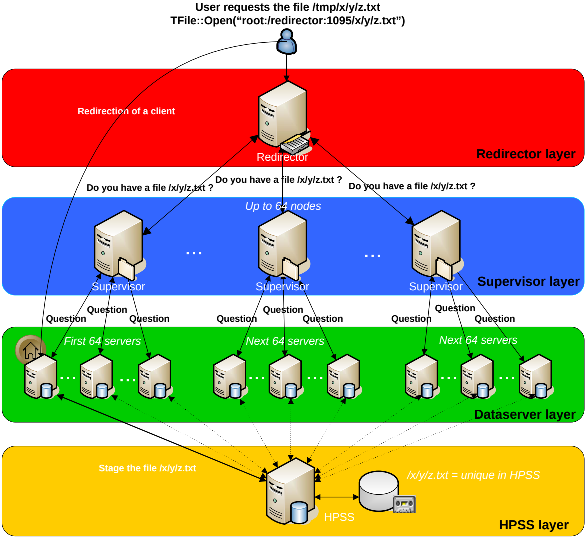
What are the required features?

- 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 - Consistent data management strategy: possibly talk to other DM tools, local or distributed on Grid
- two most popular solutions in HENP: **dCache** and **Xrootd**

Solve rootd problems with xrootd features

1. ROOTD knows only PFN ⇒ XROOTD knows "LFN"
   - data are located within xrootd process and only LFN needs to be cataloged (reducing problem from 1:N to 1:1)
2. Spidering scalability issues ⇒ XROOTD knows "LFN"
   - no need to index available data on nodes, data are located within xrootd process
3. Static data population ⇒ Mass storage system plugin
   - movement from **static** population of data to **dynamic**
   - dynamically populated disk space “**on demand**”
4. Overloaded and not responding node ⇒ Load balancing
   - xrootd determines which server is the best for client’s request to open a file
5. Job start time latency ⇒ Fault tolerance feature
   - missing data can be again restored from other storage sources

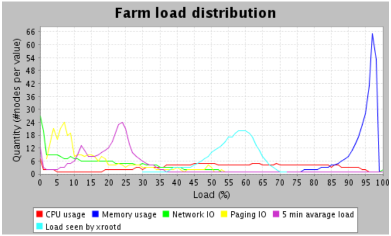
XROOTD architecture and request handling ?

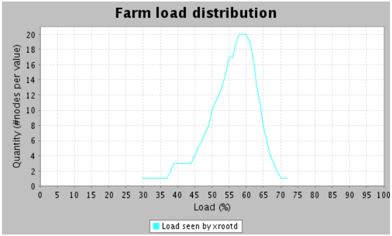


Each CE hosts SE = sharing of resource

Understanding the load to increase the performance

- distributed system can have several choices to fulfill a incoming request (more replicas of a file etc.)
- the system needs to balance the load among many collaborating servers
- xrootd offers computation of the server workload as a flexible formula:
  - it is a combination of 5 main factors (cpu, memory etc.)
- how to setup the thresholds to represent STAR's environment ?
  - is it CPU-bound, memory-bound environment?





Studying load distribution stability

- load distribution illustrated to be pretty stable over longer period of time

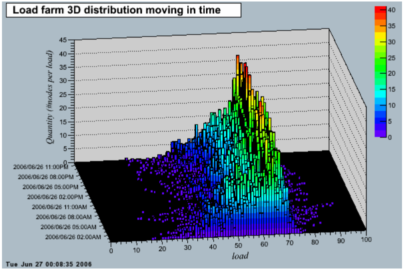


Figure: Week 26, Monday

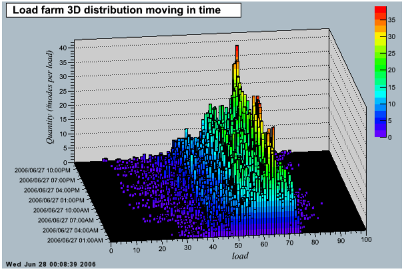
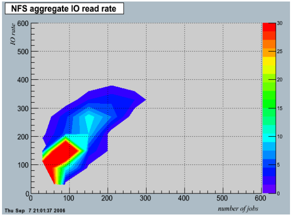


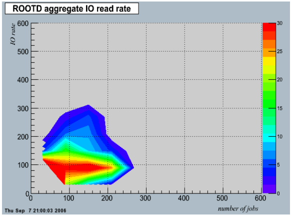
Figure: Week 26, Tuesday

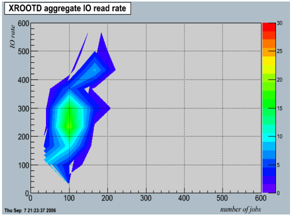
- **Question**: Does this exercise with load distribution help in terms of performance ?

Preliminary comparison of several solutions

- success of distributed file system relies on the ability to support increasing number of users with stable performance of individual file's operation
- it implies performance comparison with the following measurement's requirements:
  - relation of the aggregate IO throughput with the number of requests
  - identical measurement's conditions (same structure of files, compression etc.)



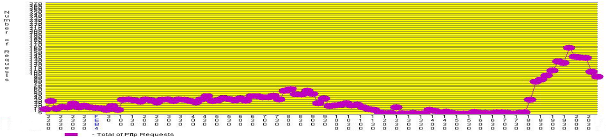




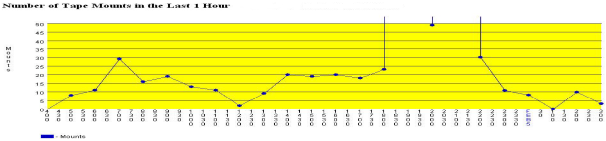
HPSS access pattern consequence

- requests to HPSS are not coordinated:
    1. increase number of requests

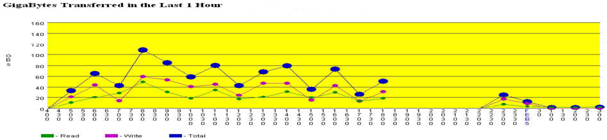
peak at 170 requests



    2. increase tape mounts to maximum



    3. decrease I/O Rate to zero

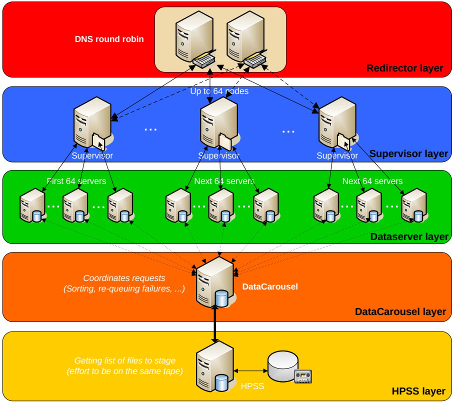


How to avoid such a collapse ?

- there is a need for a system with features as:
  - coordinating and queuing requests
  - sharing access with other data management tools involving policy based authorization with different priorities per user or group
  - keeping track of requests and re-queuing them in case of failure
  - advance techniques as: request expiration time, cancellation of request etc.
- system already exists inside STAR framework ⇒ **DataCarousel**[1]
- **Question**: How this complex system can be integrated into xrootd architecture ?

[1] *http://www.star.bnl.gov/STAR/comp/carousel/data_carousel.html*

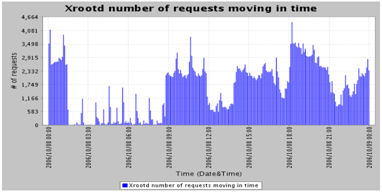
XROOTD with HPSS request coordination

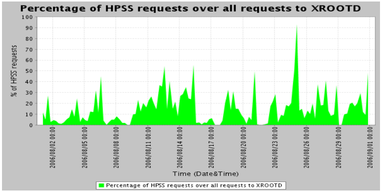


- DNS RR solution for clusters bigger than 64 servers available from January 2007 (being tested at BNL)
- ***How this widely distributed system is being monitored ?***

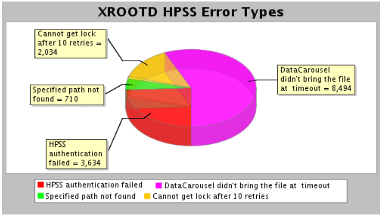
Xrootd in production and real analysis scenario

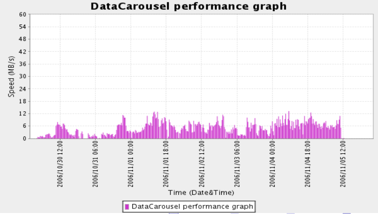
- possible to see up to 35 requests/sec to open files, users use xrootd to access HPSS data-sets





- most of errors are caused by slow performance of HPSS per tape drive (9 at STAR)





Analysis scenario (Facts at beginning)

- users defines their jobs(=analysis) using job's description of SUMS (STAR Unified Meta Scheduler)
  - SUMS resolves their meta-data query (energy, collision etc.) into particular physical data-sets by handshaking with STAR FileCatalog
  - SUMS orders data-sets, splits them into specific sub-jobs and submits into batch system queue (~100-1000 files per one sub-job)
- STAR files are compressed and structured ROOT files (events sorted into tree structure)

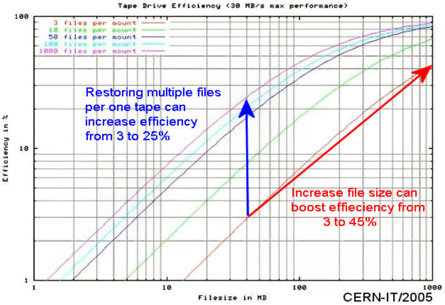
	avg size of file	file's description
an event file	~284 MB	same size as daq files
an MuDST file	~88 MB	mostly used for analysis

- our batch system has a limitation for the clock time
  - it is large, but finite and allows controlling runaway jobs
  - is this consistent with Xrootd/staging reality ?
- one file could be restored in average of 21 minutes

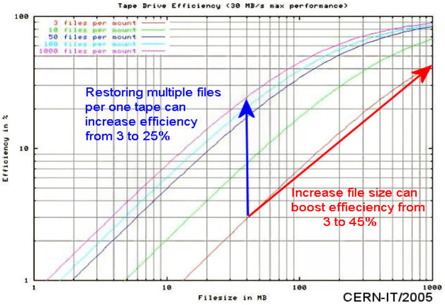
Analysis scenario (cont.)

- drawback for user's side:
  - job hangs waiting for a file to be restored from HPSS and eventually is killed by batch wall clock (e.g 21*1000=350 hours)

Reasons for slow HPSS performance

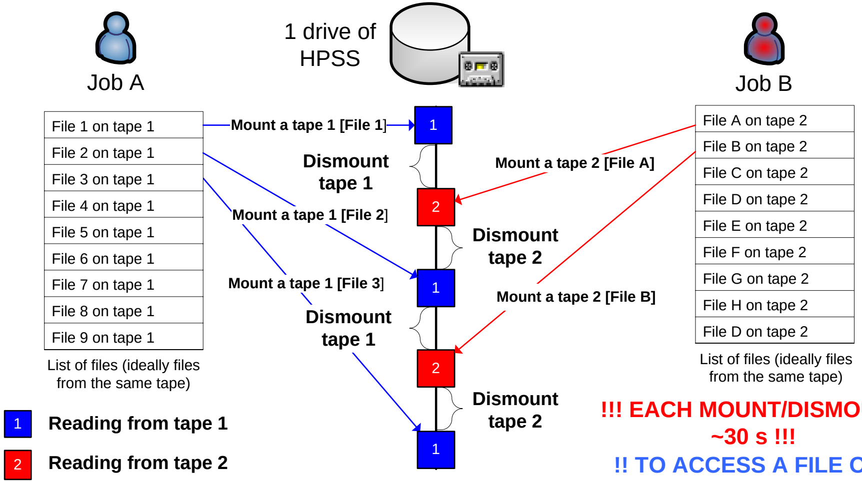


Reasons for slow HPSS performance



1. the performance can be boosted by optimizing the size of a file (currently ∼90MB)
2. sequential processing of files causes excessive mounting of tapes

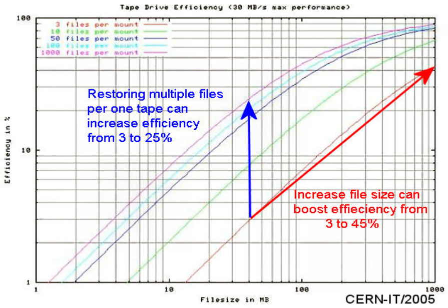
Impact of sequential processing on a HPSS drive



!!! EACH MOUNT/DISMOUNT ~30 s !!!

!! TO ACCESS A FILE ON THE TAPE ~58s !!

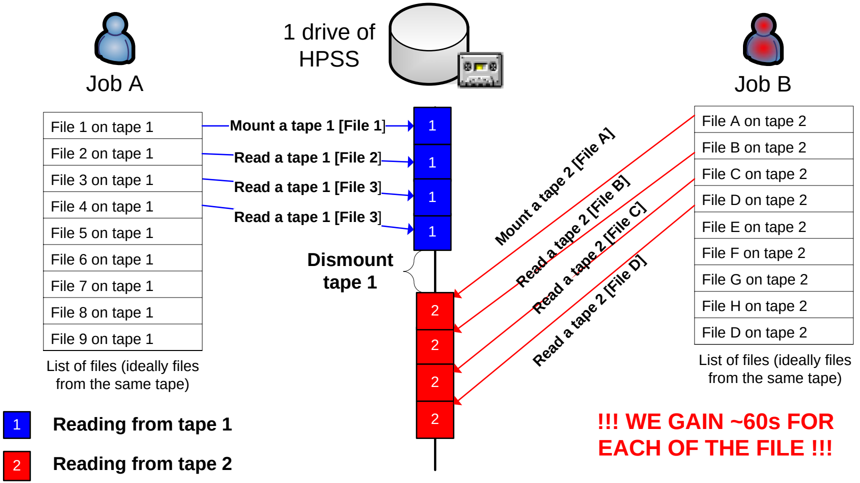
Reasons for slow HPSS performance (cont.)



1. the performance can be boosted by optimizing the size of a file (currently ~90MB)
2. sequential processing of files causes excessive mounting of tapes

- DataCarousel already does a sorting of requests per tape, but not enough efficient in case of sequential processing
- we need bigger list for sorting ⇒ Pre-staging of files
  - job publishes its whole intend for processing ⇒ all files to be processed in near future
  - files from the same tape will end up in DataCarousel system ⇒ bigger list for selection

Influence of the pre-staging on a HPSS drive



!!! WE GAIN ~60s FOR EACH OF THE FILE !!!

How the pre-staging works ?

- users have to add specific command before macro processing in their job's description

  /star/u/starlib/ROOT/xrootd/bin/preStageList ${FILELIST}

- script publishes whole list to the head node of the xrootd cluster
  - each file's presence on the cluster is investigated
  - missing files are scheduled to retrieve them from HPSS
  - everything runs at the background and is handled by server's side of xrootd ⇒ pre-Staging doesn't give an overhead to job's run-time
  - job processing is executed immediately after the publishing
- implementation details:
  - script is a wrapper around xrootd client libraries and pass files through the "prepare" protocol to the server's side
  - pre-Staging doubled a load on the head node ⇒ large file lists made head node inaccessible
  - concurrency check and subsequent flushing were added to decrease the load ⇒ script runs little bit longer

Did the pre-staging help ?

50% of improvement

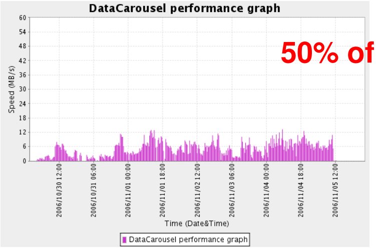


Figure: Before

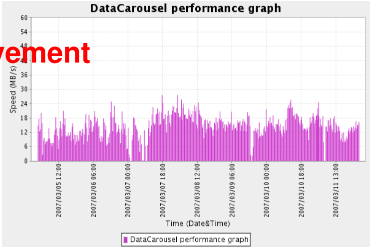
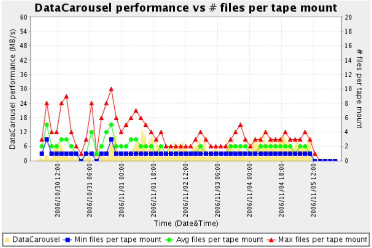
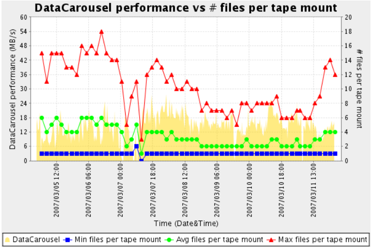


Figure: After





Motivation . . .

XROOTD has its own shortcomings 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 middle-ware are designed for space management

Leveraging on other projects and targeted re-usable components ?

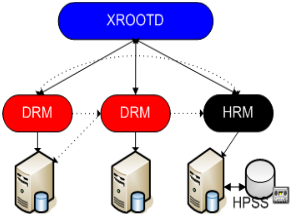
SRM functionality

- **SRM:** the grid middle-ware 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 overview

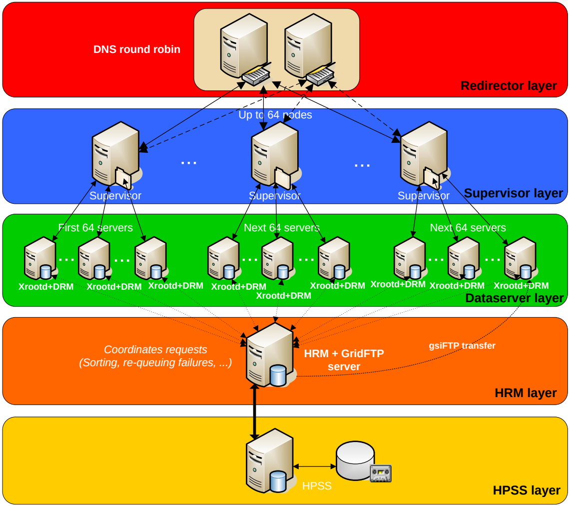
Types of storage resource managers:

- 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



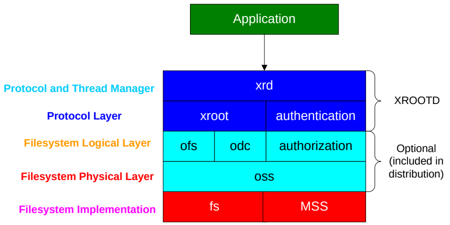
- xrootd is responsible for managing the disk cluster (aggregation, load balancing . . .)
- DRM is responsible for managing the disk cache
- HRM is responsible for managing access to HPSS

XROOTD+SRM cluster overview



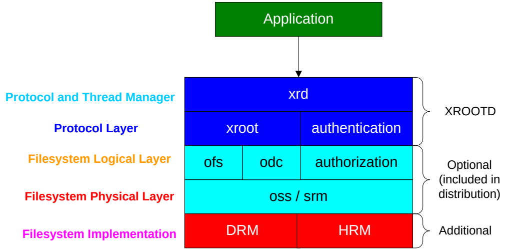
XROOTD components architecture

- xrootd architecture is very amenable to extensions (divided into several components and plug-ins)



- xrd - provides networking support, thread management and protocol scheduling
- ofs - provides enhanced first level access to file data (responsible for coordinating activities of oss, odc, auth)
- oss - provides access to underlying storage system (controlled by ofs and invokes meta-data operations)

XROOTD+SRM components architecture



- oss component was externalized as a plugin
- several existing methods were virtualized (Create, Open, Close, Stage)
  - easy wrapping of concrete class implementation (for example: calling HRM client API)
- for example:
  - Create() uses DRM to create a file
  - Close() informs that the file is no longer in use

Current status ?

- The xrootd-SRM activity is a collaboration between:
  - BNL/STAR - proposed the idea, providing early demonstrations, performing measurement, provide enhancements
  - SLAC - providing changes to xrootd, providing distribution mechanism with xrootd
  - LBNL/SDM - providing plug-ins that interact with SRMs and changes to DRM for new functionality
- working version:
  - supports read, write mode into HPSS with all SRM functionalities as pinning, allocation etc.
    - thanks to A. Romosan and A. Sim
  - supports one cache per a node $\Rightarrow$ multiple caches per one node has been recently developed (under testing)
  - planning to test it in large scale at BNL after multiple caches support and later production mode

Summary

- Xrootd is currently deployed on almost 500 nodes (the biggest production deployment of xrootd) serving over 350TB of distributed disk space
- load balancing and handshake with tape system make the system resilient to failures
- the system is used by users for daily analysis
- measurement of xrootd aggregate IO showed competitive results comparing to commercial Panasas(NFS)
- script for Pre-staging will be replaced with new ROOT class *TFileStager*
- future plans:
  - large scale testing of SRM+XROOTD is on the way
  - planning a regression tests (performance comparison with previous version)
  - planning enhancements such as cache-to-cache transfers and interoperability with other SRM-aware tools (such as DataMovers etc.)

References

P. Jakl, J. Lauret, A. Hanushevsky, A. Shoshani, A. Sim
From rootd to xrootd: From physical to logical file
Proc. of Computing in High Energy and Nuclear Physics (CHEP'06)

P. Jakl, J. Lauret, M. Šumbera
Managing widely distributed data–sets
Czech Technical University, FNSPE, Research report, 2006
http://www.star.bnl.gov/~pjakl

A. Romosan, A. Hanushevsky
XROOTD–SRM
SRM–Collaboration meeting, CERN, 2006