Scalla/xrootd

Andrew Hanushevsky, SLAC

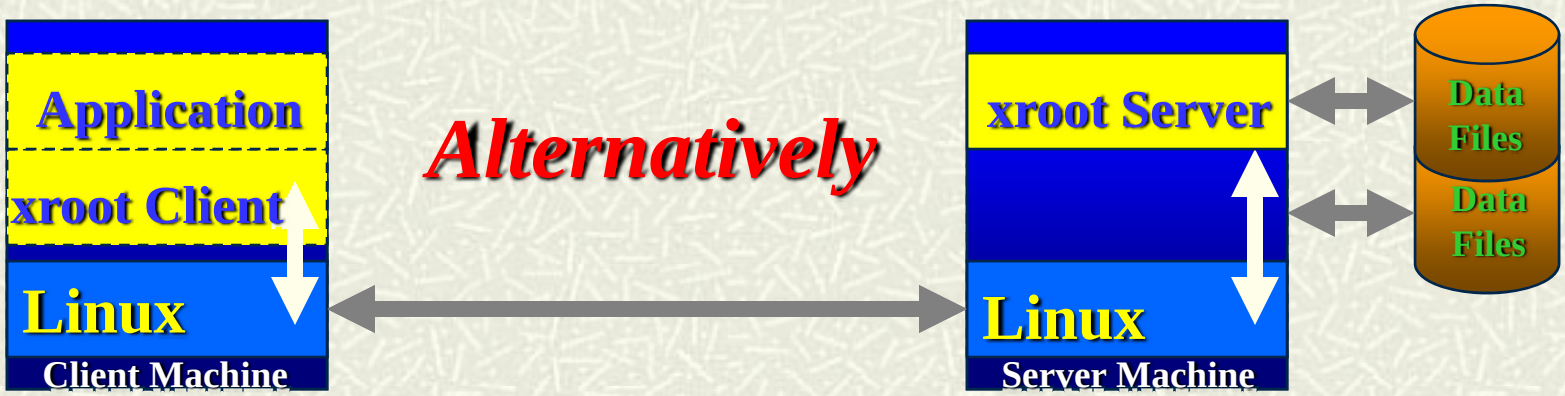
SLAC National Accelerator Laboratory
Stanford University
19-May-09

ANL Tier3(g,w) Meeting

Outline

- File servers
  - NFS & xrootd
- How xrootd manages files
  - Multiple file servers (i.e., clustering)
  - Considerations and pitfalls
- Getting to xrootd hosted file data
- Native monitoring
- Conclusions

File Server Types

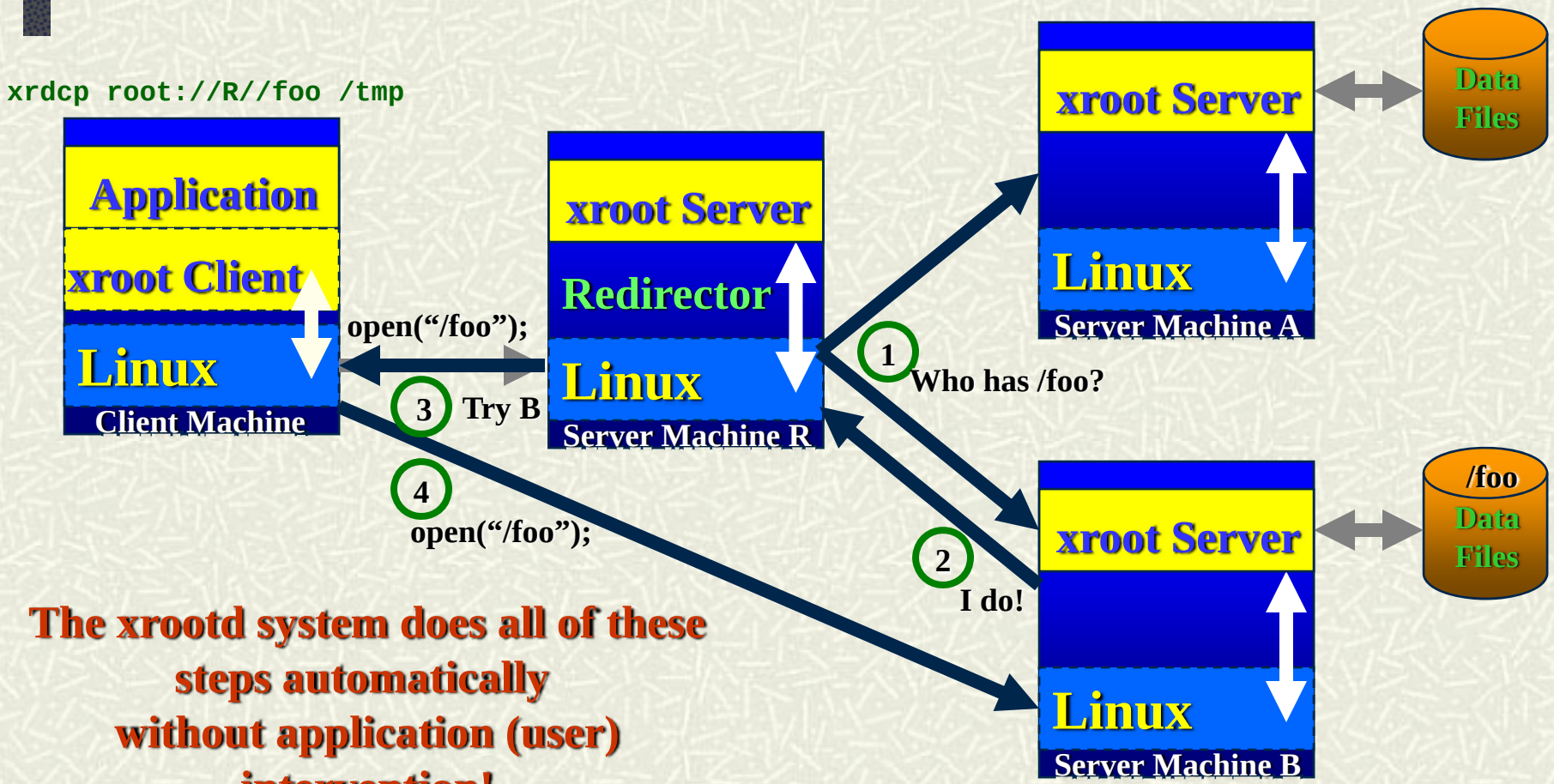


xrootd is nothing more than an *application level file server & client* using another protocol

Why Not Just Use NFS?

- NFS V2 & V3 inadequate
  - Scaling problems with large batch farms
  - Unwieldy when more than one server needed
- NFS V4?
  - Relatively new
    - Standard is still being evolved
      - Mostly in the area of new features
    - Multiple server clustering & stress stability being vetted
  - Performance appears similar to NFS V3
- Let's explore multiple server support in xrootd

xrootd & Multiple File Servers I



The xrootd system does all of these steps automatically without application (user) intervention!

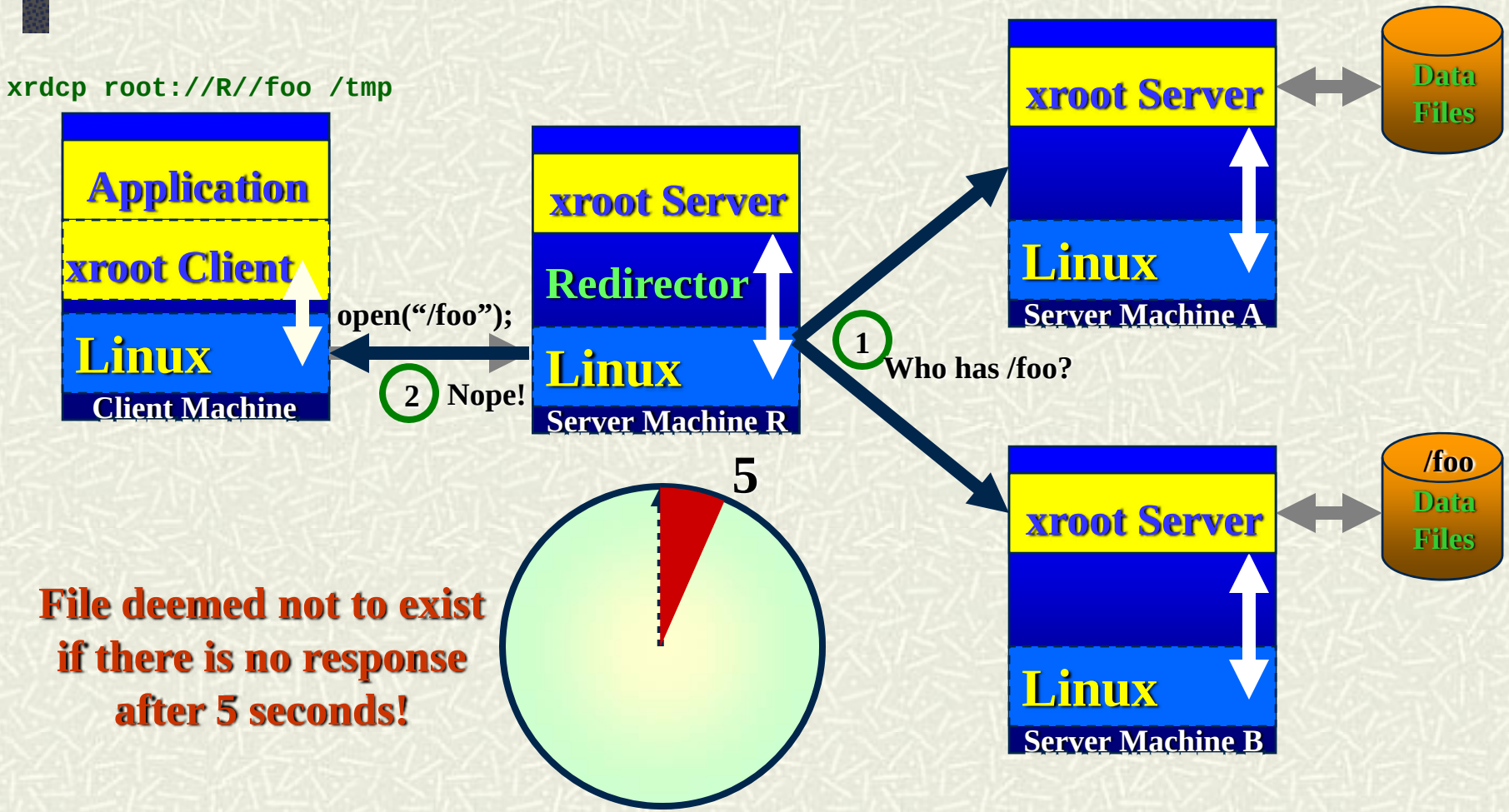
Corresponding Configuration File

```
# General section that applies to all servers
#
all.export /atlas

if redirector.slac.stanford.edu
all.role manager
else
all.role server
fi
all.manager redirector.slac.stanford.edu 3121

# Cluster management specific configuration
#
cms.allow *.slac.stanford.edu

# xrootd specific configuration
#
xrootd.fslib /opt/xrootd/prod/lib/libXrdOfs.so
xrootd.port 1094
```

File Discovery Considerations I

- The redirector does not have a catalog of files
  - It always asks each server, and
  - Caches the answers in memory for a “while”
    - So, it won’t ask again when asked about a past lookup
- Allows real-time configuration changes
  - Clients never see the disruption
- Does have some side-effects
  - The lookup takes less than a millisecond when files exist
  - Much longer when a requested file does not exist!

xrootd & Multiple File Servers II



File Discovery Considerations II

- System optimized for “file exists” case!
  - Penalty for going after missing files
- Aren’t *new* files, by definition, missing?
  - Yes, but that involves writing data!
    - The system is optimized for reading data
  - So, creating a new file *will* suffer a 5 second delay
    - Can minimize the delay by using the **xprep** command
      - Primes the redirector’s file memory cache ahead of time
- Can files appear to be missing any other way?

Missing File vs. Missing Server

- In xrootd files exist to the extent servers exist
  - The redirector cushions this effect for 10 minutes
    - The time is configurable, but…
      - Afterwards, the redirector cannot tell the difference
  - This allows partially dead server clusters to continue
    - Jobs hunting for “missing” files will eventually die
    - *But* jobs cannot rely on files actually being missing
      - xrootd cannot provide a definitive answer to **“∀ s: ¬∃ file x”**
- This requires additional care during file creation
  - Issue will be mitigated in next release
    - Files that persist only when successfully closed

Getting to xrootd hosted data

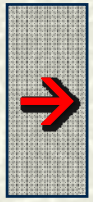
- Via the root framework
  - Automatic when files named root://....
  - Manually, use TXNetFile() object
    - Note: identical TFile() object will not work with xrootd!
- xrdcp
  - The native copy command
- SRM (optional add-on)
  - srmcp, gridFTP
- FUSE
  - Linux only: xrootd as a mounted file system
- POSIX preload library
  - Allows POSIX compliant applications to use xrootd

The Flip Side of Things

- File management is largely transparent
  - Engineered to be turned on and pretty much forget
- But what if you just need to know
  - Usage statistics
    - Who's using what
    - Specific data access patterns
  - The big picture
    - A multi-site view

Xrootd Monitoring Approach

- Minimal impact on client requests
- Robustness against multimode failure
- Precision & specificity of collected data
- Real time scalability



Use UDP datagrams

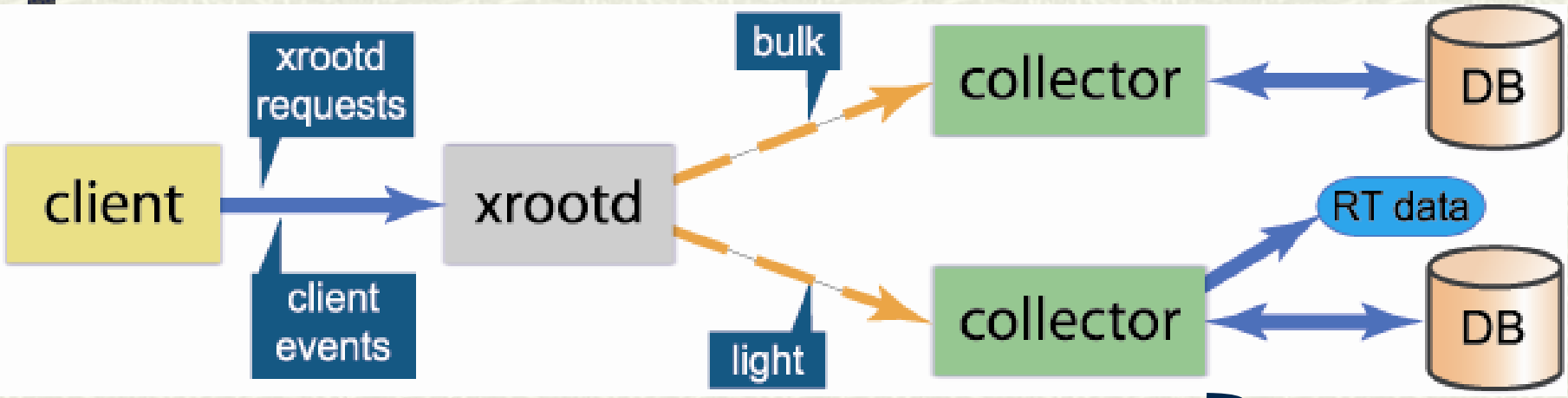
- Data servers insulated from monitoring. But
- Packets can get lost

Highly encode the data stream

Outsource stream serialization

Use variable time buckets

Monitored Data Flow

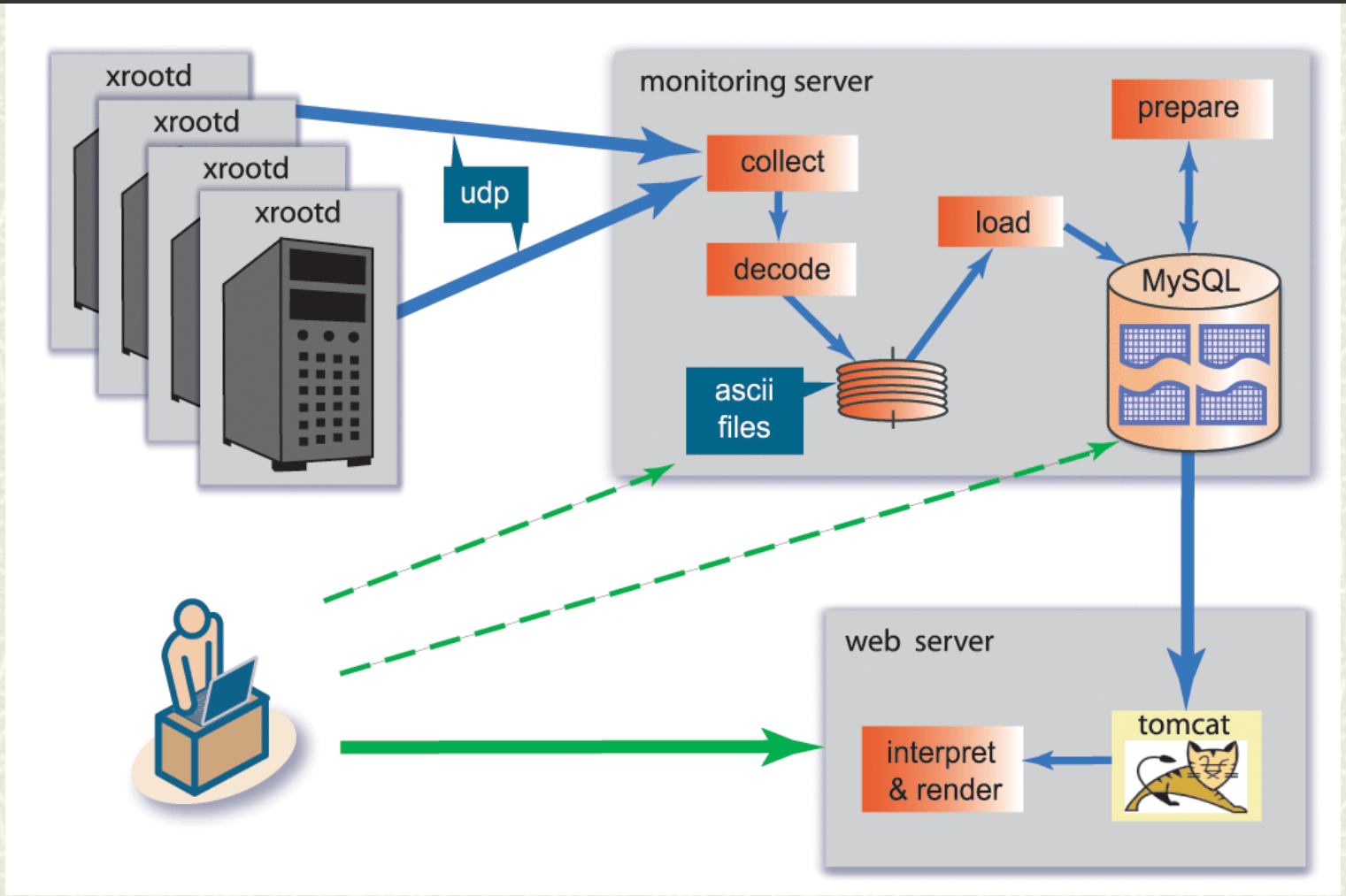


RT data

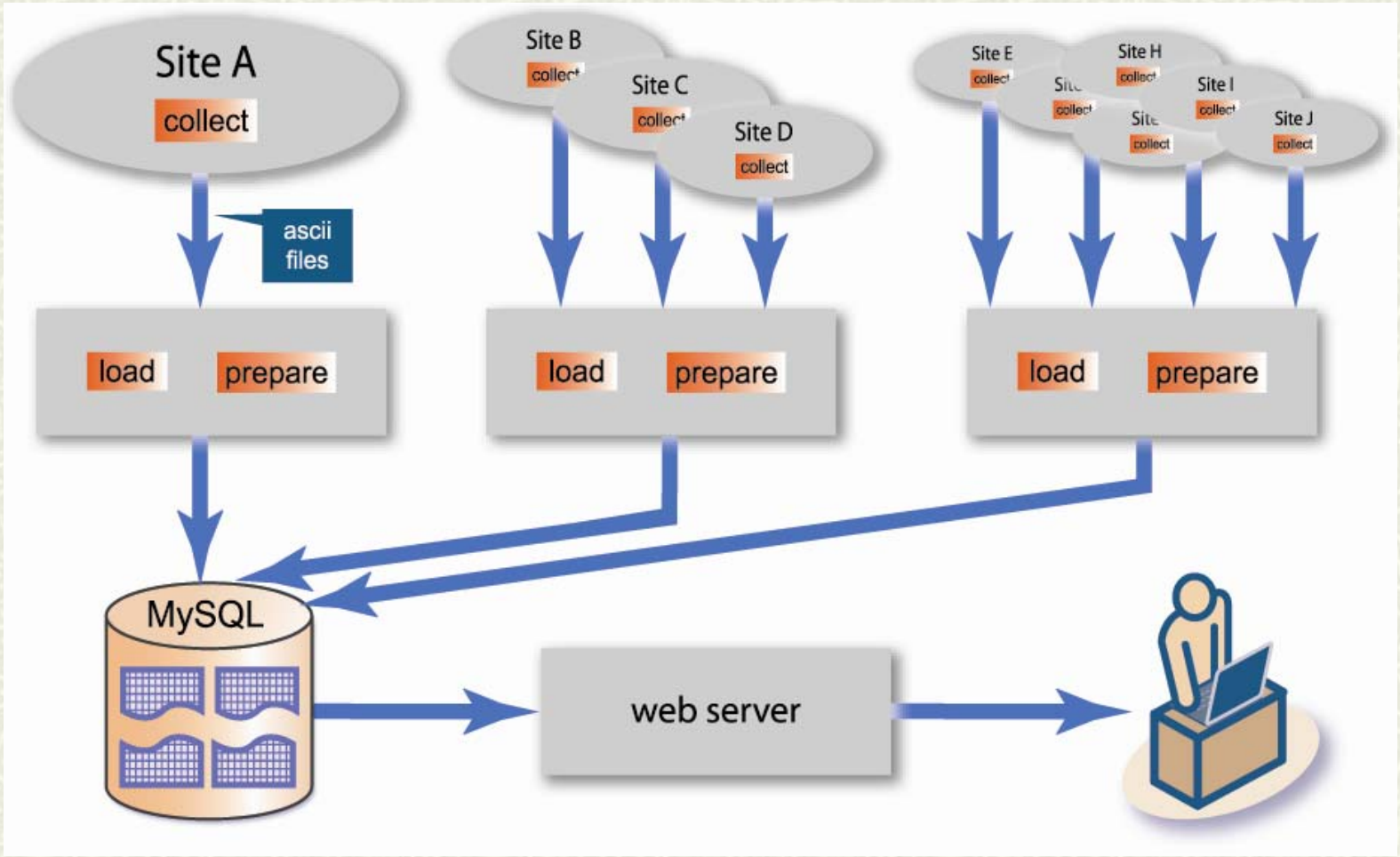
- **Start Session**
  - sessionId, user, PId, client, server, timestamp
- **Open File**
  - sessionId, fileId, file path, timestamp
- **Bulk I/O**
  - sessionId, fileId, file offset, number bytes
- **Close File**
  - sessionId, fileId, bytes read, bytes written
- **Application Data**
  - sessionId, appdata
- **End Session**
  - sessionId, duration, server restart time
- **Staging**
  - stageId, user, PId, client, file path, timestamp, size, duration, server

Configurable

Single Site Monitoring

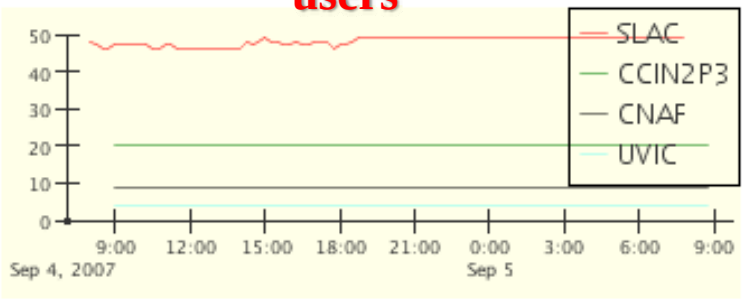


Multi-Site Monitoring

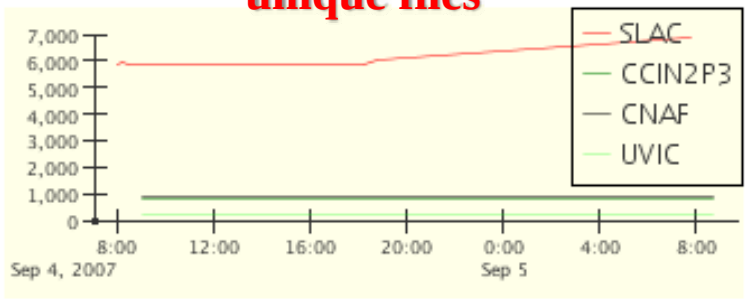


Basic Views

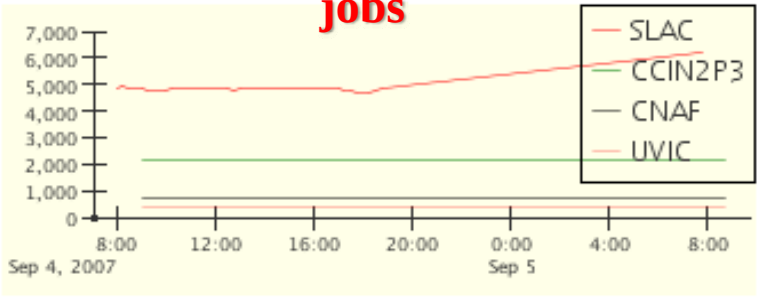
users



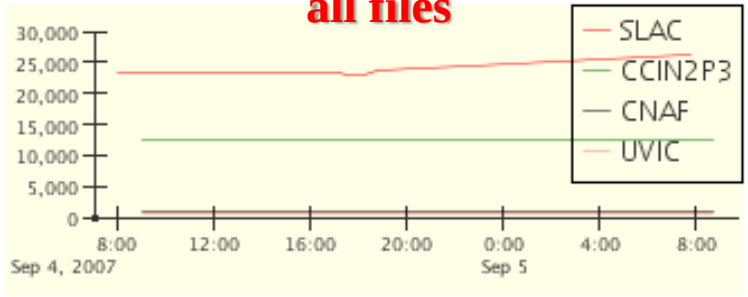
unique files



jobs



all files



Detailed Views

Top Performers Table

Table rows: 5 | Time Period: Last Hour | Site: SLAC | Update

Top active users

User Name	Now			Last Hour			
	Number of Jobs	Number of Files	File Size [MB]	Number of Jobs	Number of Files	File Size [MB]	MB Read
ayarritu	615	139	65,987	430	146	65,802	41,360
jregens	360	405	371,874	64	317	303,252	143,852
cschill	281	32	27,133	79	30	25,301	4,892
feltresi	149	106	167,528	70	143	218,873	74,552
torsten	72	99	83,673	184	1,532	630,092	235,327

Hottest dataTypes

dataType Name	Now				Last Hour				
	Number of Jobs	Number of Files	File Size [MB]	Number of Users	Number of Jobs	Number of Files	File Size [MB]	Number of Users	MB Read
SPskims	998	739	632,651	11	663	340	304,938	6	120,728
SP	652	1,839	1,961,610	12	981	506	474,819	7	159,512
PRskims	93	650	811,152	7	204	83	107,807	2	62,265
PR	66	600	453,640	6	265	1,454	525,498	3	174,754
cfg	0	0	0	0	8	1	7	1	10

Hottest skims

skim Name	Now				Last Hour				
	Number of Jobs	Number of Files	File Size [MB]	Number of Users	Number of Jobs	Number of Files	File Size [MB]	Number of Users	MB Read
BtoRhoGamma	591	139	65,987	1	458	146	65,802	1	41,360
DstToD0PiToVGamma	262	86	33,138	1	70	41	16,171	1	4,668
BToDlnu	115	118	186,026	2	125	145	222,200	2	74,568
AllEvents	76	394	508,309	3	210	84	108,365	3	62,268
Tau11	4	95	130,103	1	3	6	149	0	127

Hottest files

File Path	File Size [MB]	Now	Last Hour	
		Number of Jobs	Number of Jobs	MB Read
/store/PRskims/R18/18.6.3d/AllEvents/00/AllEvents_20006.04HB.root	1,690	2	15	1,630
/store/PRskims/R18/18.6.3e/AllEvents/05/AllEvents_20502.04HB.root	1,688	1	17	1,636
/store/PRskims/R18/18.6.3e/AllEvents/05/AllEvents_20502.01.root	1,689	1	17	1,635
/store/PRskims/R18/18.6.3e/AllEvents/05/AllEvents_20500.03HB.root	1,688	1	19	1,641
/store/PRskims/R18/18.6.3e/AllEvents/05/AllEvents_20500.01.root	1,689	1	19	1,640

Per User Views

User Information

Now		Last Hour	
Number of Running Jobs	203	Number of Finished Jobs	831
		Total Duration of all Jobs [DAY HH:MM:SS]	74 16:46:57
Number of Open Sessions	388	Number of Closed Sessions	1,865
Number of Open Files	146	Number of Accessed files	1,241
		Volume of Data Read [MB]	719,109
		Volume of Data Written [MB]	0
Number of Client Hosts in Use	157	Number of Client Hosts Used	593
Number of Server Hosts in Use	44	Number of Server Hosts Used	50

What's Missing

- Integration with common tools
  - Nagios, Ganglia, MonaLisa, etc.
- Better Packaging
  - Simple install
- Better Documentation
- Working on proposal to address the issues

The Good Part I

- Xrootd is simple and easy to administer
  - E.g.: BNL/Star 400-node cluster → 0.5 grad student
  - No 3rd party software required (i.e., self-contained)
    - Not true when SRM support needed
  - Single configuration file independent of cluster size
- Handles heavy unpredictable loads
  - E,g., >3,000 connections & >10,000 open files
    - Ideal for batch farms where jobs can start in waves
- Resilient and forgiving
  - Configuration changes can be done in real time
    - Ad hoc addition and removal of servers or files

The Good Part II

- Ultra low overhead
  - Xrootd memory footprint < 50MB
    - For mostly read-only configuration on SLC4 or later
    - Opens a wide range of deployment options
- High performance LAN/WAN I/O
  - CPU overlapped I/O buffering and I/O pipelining
    - Well integrated into the root framework
    - Makes WAN random I/O a realistic option
  - Parallel streams and optional multiple data sources
    - Torrent-style WAN data transfer

The Good Part III

- Wide range of clustering options
  - Can cluster geographically distributed clusters
  - Clusters can be overlaid
    - Can run multiple xrootd versions using production data
- SRM V2 Support
  - Optional add-on using LBNL BestMan
- Can be mounted as a file system
  - FUSE (SLC4 or later)
    - Not suitable for high performance I/O
- Extensive monitoring facilities

The Not So Good

- Not a general all-purpose solution
  - Engineered primarily for data analysis
  - Not a true full-fledged file system
    - Non-transactional file namespace operations
      - Create, remove, rename, etc
        - Create mitigated in the next release via ephemeral files
- SRM support not natively integrated
  - Yes, 3rd party package
- Too much reference-like documentation
  - More tutorials would help

Conclusion

- Xrootd is a lightweight data access system
  - Suitable for resource constrained environments
    - Human as well as hardware
  - Rugged enough to scale to large installations
    - CERN analysis & reconstruction farms
- Readily available
  - Distributed as part of the OSG VDT
    - Also part of the CERN root distribution
- Visit the web site for more information
  - http://xrootd.slac.stanford.edu/

Acknowledgements

- Software Contributors
  - Alice: Derek Feichtinger
  - CERN: Fabrizio Furano , Andreas Peters
  - Fermi/GLAST: Tony Johnson (Java)
  - Root: Gerri Ganis, Beterand Bellenet, Fons Rademakers
  - SLAC: Tofigh Azemoon, Jacek Becla, Andrew Hanushevsky, Wilko Kroeger
  - LBNL: Alex Sim, Junmin Gu, Vijaya Natarajan (BeStMan team)
- Operational Collaborators
  - BNL, CERN, FZK, IN2P3, RAL, SLAC, UVIC, UTA
- Partial Funding
  - US Department of Energy
    - Contract DE-AC02-76SF00515 with Stanford University