BERGISCHE
UNIVERSITÄT
WUPPERTAL

XRootD

Sergey Kalinin

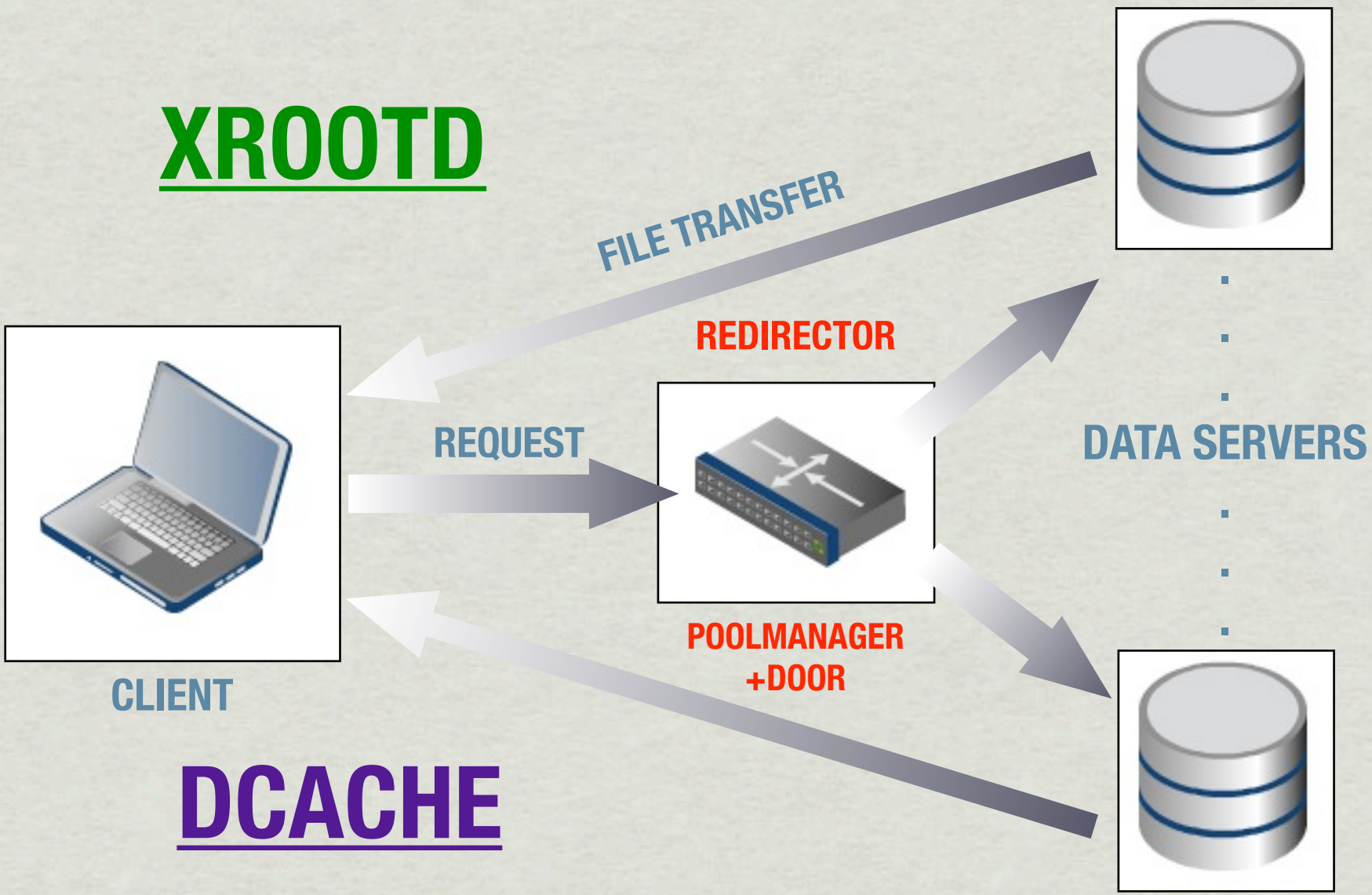
Outline

- Short architecture description
- **dCache** as an **XRootD** server
  - Configuration
  - Security
- Results of performance tests

XRootD

- A data transfer protocol(e.g. ‘xrootd://’ or ‘root://’)
- Storage system
  - High performance scalable data access
  - Storage aggregation(disks/machines/sites)
  - Plugin based
  - File system is used for namespace. No databases.

Architecture overview



dCache as XRootD server

As simple as adding an XRootD door in

`${dCacheHome}/etc/layouts/head.conf` :

```
..
[xrootd-${host.name}Domain]
[xrootd-${host.name}Domain/xrootd]
..
```

STOLEN FROM THE BOOK!

Restart now the domain and the door will be there. You can access files with **xrdcp** or **TFile::Open("root://site.name/pnfs/...")**.

Since 1.9.10 TCP ports are re-used and number of parallel transfers is not limited anymore by TCP port range.

Security

XRootD uses libraries for authentication/authorization but **dCache** uses its own plugins for this. There are several ways to configure **XRootD** access security:

- Read-Write access for the complete namespace of just for some directories
- Token-based authorization
- Strong authentication

All of them are very well described in the Book

http://www.dcache.org/manuals/Book-1.9.11/config/cf-xrootd-setup.shtml

But remember that the same authorization mechanism is used for all protocols except that token-based works only for xrootd(e.g. Alice).

Read-write access.

To enable read-**AND**-write access ad the following to

`${dCacheHome}/etc/dcache.conf` :

```
..
xrootdAllowedPaths=/pnfs/<example.org>/path1:/pnfs/<example.org>/path2
..
```

If you want read-**OR**-write :

```
..
xrootdAllowedReadPaths=/pnfs/<example.org>/rpath1:/pnfs/<example.org>/rpath2
xrootdAllowedWritePaths=/pnfs/<example.org>/wpath1:/pnfs/<example.org>/wpath2
..
```

Token-based authorization(ALICE)

As suggested in http://people.web.psi.ch/feichtinger/doc/authz.pdf

Generate new RSA private key

```
[root] # openssl genrsa -rand 12938467 -out key.pem 1024
```

Create certificate request

```
[root] # openssl req -new -inform PEM -key key.pem -outform PEM -out certreq.pem
```

Create certificate by self-signing certificate request

```
[root] # openssl x509 -days 3650 -signkey key.pem -in certreq.pem -req -out cert.pem
```

Extract public key from certificate

```
[root] # openssl x509 -pubkey -in cert.pem -out pkey.pem
[root] # openssl pkcs8 -in key.pem -topk8 -nocrypt -outform DER -out
<new_private_key>
[root] # openssl enc -base64 -d -in pkey.pem -out <new_public_key>
```

and make dCache aware of this key in **${dCacheHome}/etc/dcache.conf**

```
..
xrootdAuthzPlugin=org.dcache.xrootd.security.plugins.tokenauthz.TokenAuthorizationFactory
xrootdAuthzKeystore=<Path_to_your_Keystore>
..
```

Strong authentication.

To enable GSI authentication, add the following to **`${dCacheHome}/etc/dcache.conf`**:

```
..
xrootdAuthNPlugin=gsi
verifyHostCertificateChain=true
..
```

Note

The `xrootd`-door can be configured to use either token authorization or strong authentication with `gPlazma` authorization. A combination of both is currently not possible.

OS level tuning for data servers

- Linux defaults are usually not optimal for data servers!
- TCP buffer size in **/etc/sysctl.conf**

```
net.core.rmem_max = 8388608
net.core.wmem_max = 8388608
net.ipv4.tcp_rmem = 4096 87380 8388608
net.ipv4.tcp_wmem = 4096 65536 8388608
net.core.netdev_max_backlog = 250000
```

- TX queue length of network interfaces

```
ifconfig eth2 txqueuelen 50000
```

- Read ahead

```
/sbin/blockdev --setra 8192 /dev/cciss/c1d0p1
```

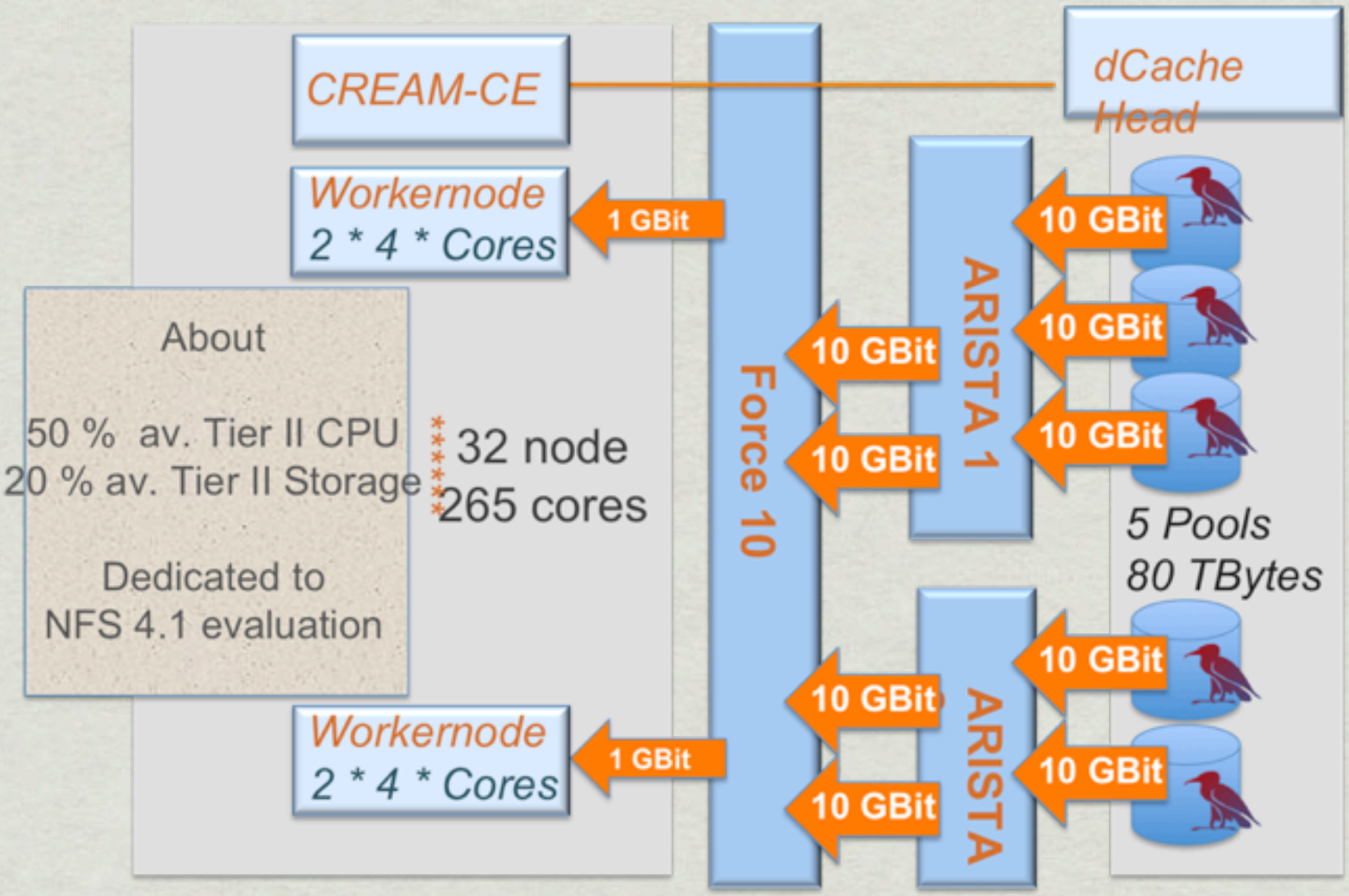
- RAID scheduler queue

```
echo noop > /sys/block/cciss\!c1d0/queue/scheduler
```

Performance and stability

- Server side improvements
  - JBoss Netty(a framework for asynchronous event-driven network applications)
  - Java native IO for disks operations
  - The same TCP port can handle different transfers since 1.9.10!
- Server provided IO mechanisms
  - Blocks read ahead
  - Vector read for ROOT files

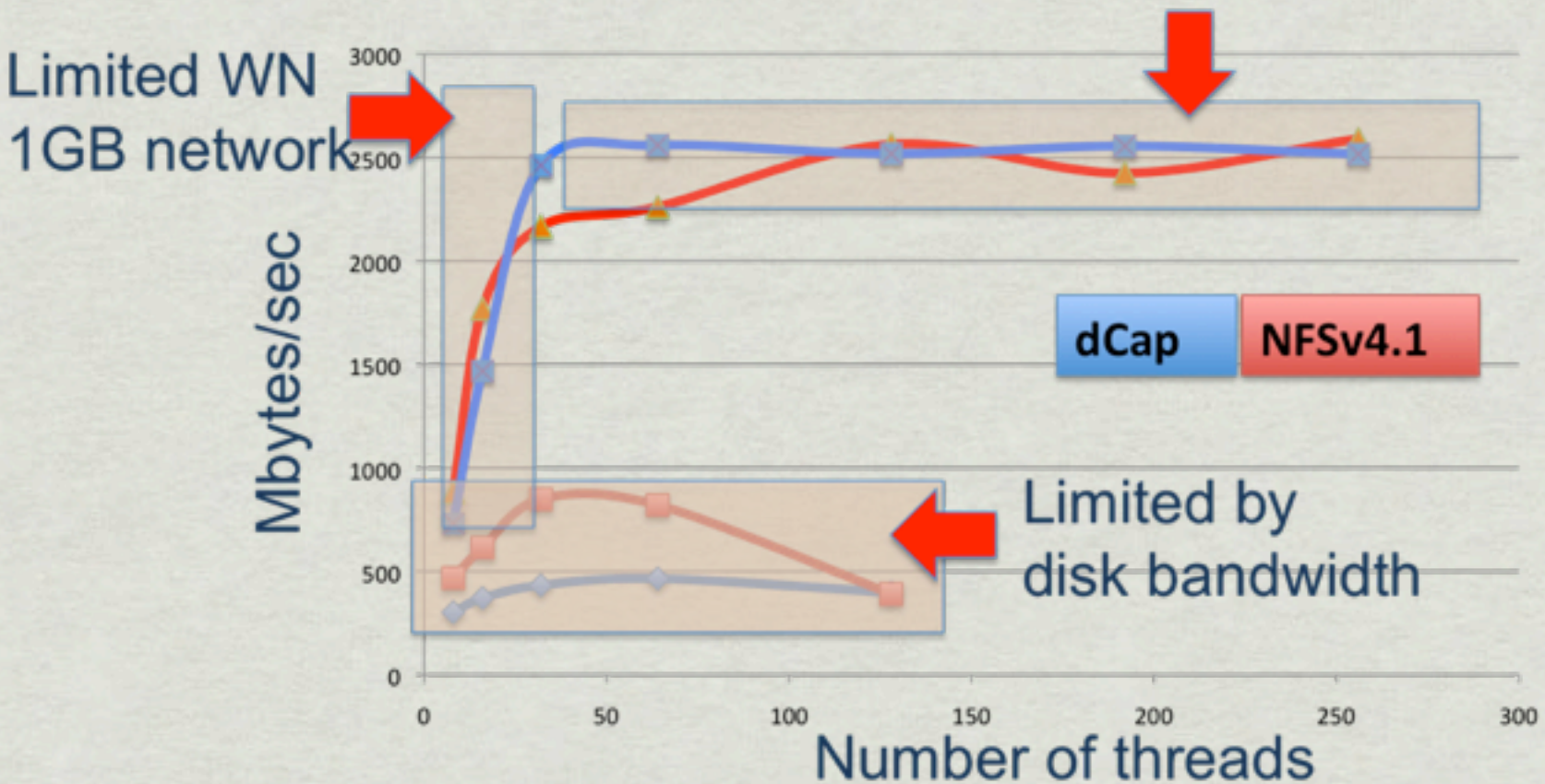
Test system setup. Hardware.



Test system limits at GridLab, DESY-HH

THE FILES ARE KEPT IN MEMORY USING HARD LINKS!

Removing server disk congestion effect by keeping all data in file system cache of the pool.



Test system setup. Software.

- The idea was to test **dCache/dcap**, **dCache/xrootd**, **dCache/nfs4.1** and original **SLAC XRootD/xrootd**
- The script which reads data comes from René Brun who uses it for his tests.
- No special tuning for dCache. SLAC XRootD required switching off debug mode as we were told by an XRootD expert from CERN.
- XRootD files were symlinks to dCache files located on the same machine.

Tests description

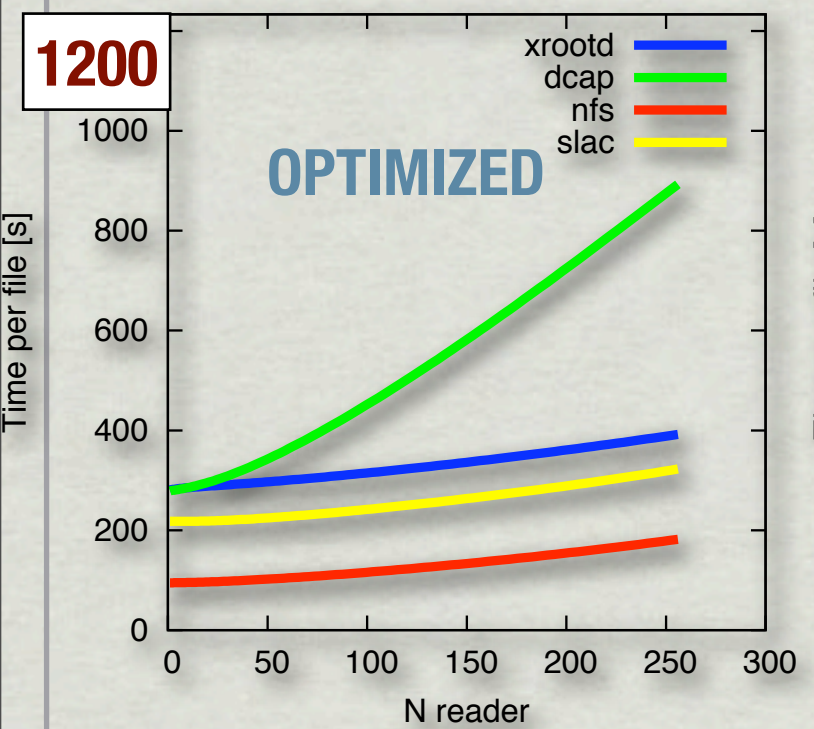
Test	Description
Not optimized	for reading- ATLAS AODs without basket ordering
Optimized	Similar but with basket ordering optimized for fast reading. The content and files sizes are different compared to not optimized
TTreeCache size	ROOT's own read ahead cache using vector read for IO:0 or 60 MB
# of Branches	Read ALL(~13) branches or just TWO

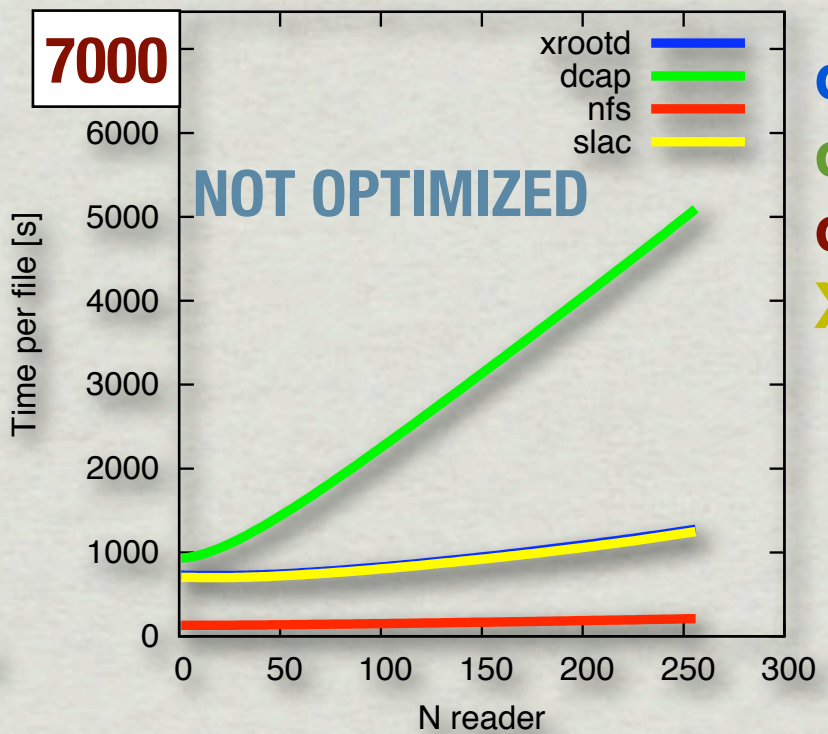
dcap results are not up-to-date

- Unfortunately, dCache/dcap results should be disregarded because the client library did not contain dcap++ buffering updates from Günter Duckeck

Reading all branches

- Optimized is much faster than not optimized!
- XROOTD in dCache is getting closer to original SLAC XRootD
- NFS4.1 seems to be better likely because of file system caching





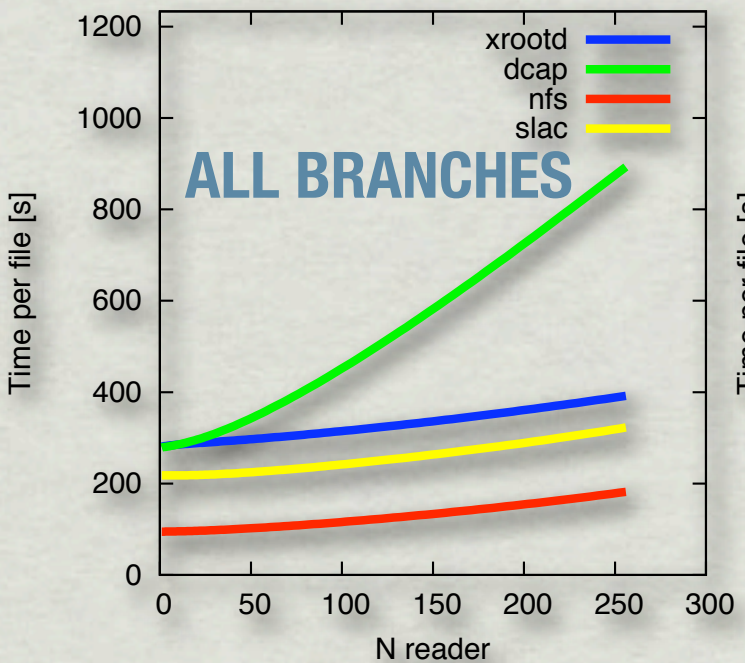
dCache/xrootd
dCache/dcap
dCache/nfs4.1
XRootD/xrootd

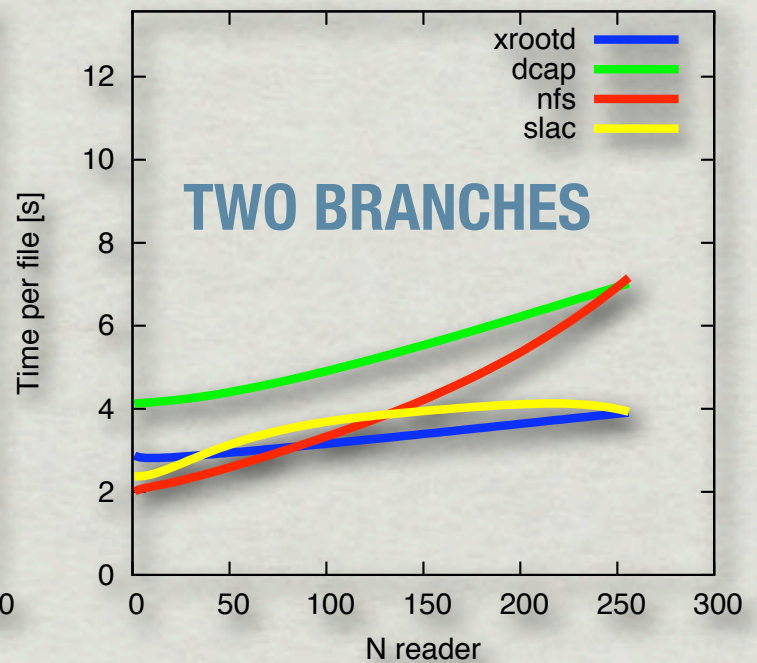
Test details: TTreeCache 0 B!
Location: GridLab, DESY-HH

Reading a subset of branches(optimized)

'Two branches' test represents a case when you need only a fraction of data presented in file.

- What is the best for you depends on what you do.
- Performance highly depends on access profile



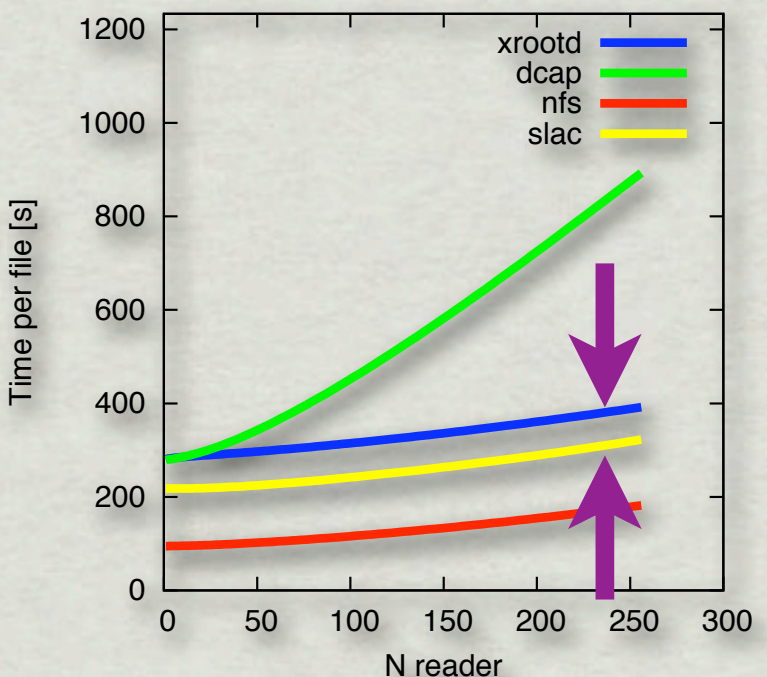


dCache/xrootd
dCache/dcap
dCache/nfs4.1
XRootD/xrootd

Test details: TTreeCache 0 B!
Location: GridLab, DESY-HH

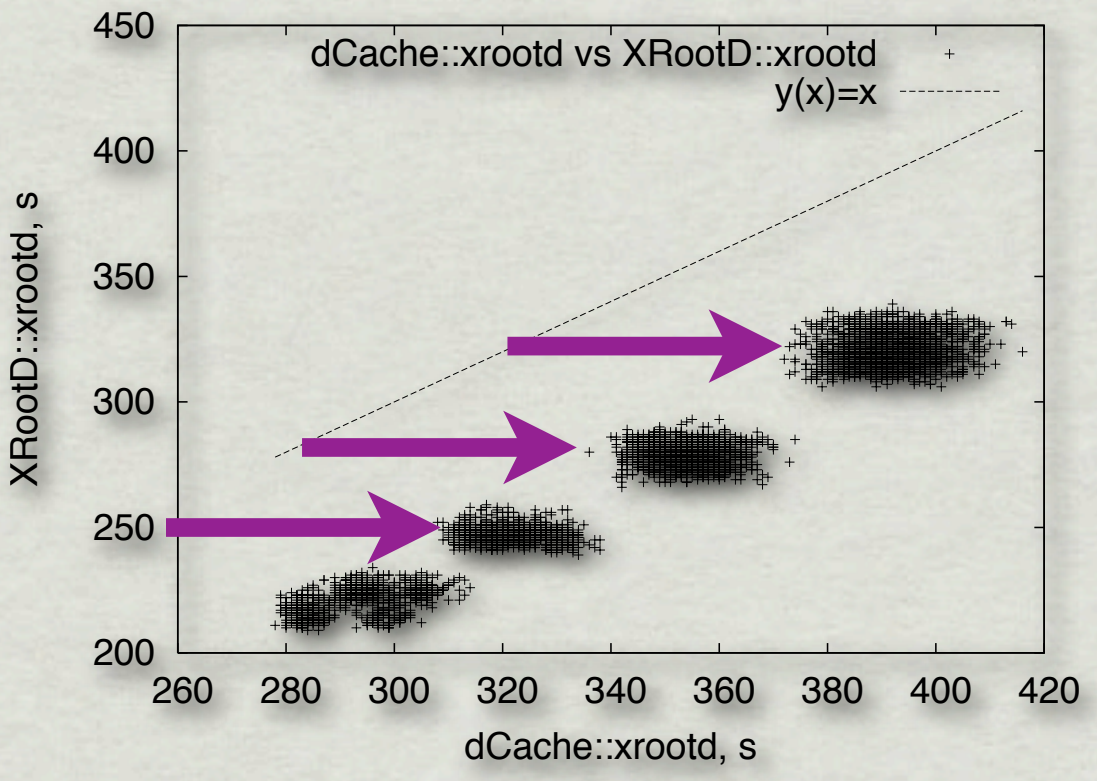
dCache/xrootd vs XRootD/xrootd

Two similar instances were tested: one is dCache and the other one is original SLAC/XRootD daemon. The shift is not yet understood but known. Investigation is going on.



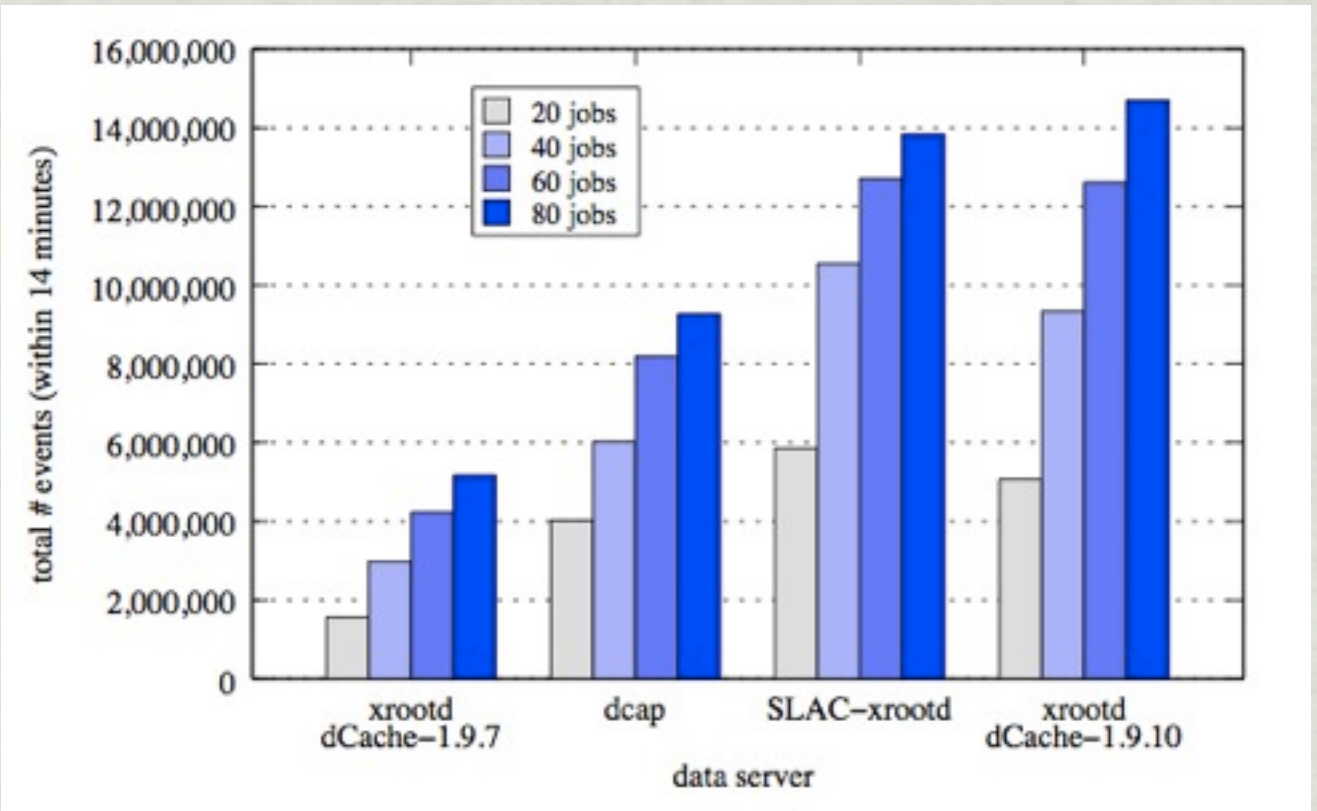
Test details: TTreeCache 0 B!

Location: GridLab, DESY-HH



HEPiX group tests

- There are significant improvements in xrootd implementation between 1.9.7 and 1.9.10
- SLAC/xrootd and dCache/xrootd in 1.9.10 have comparable performances

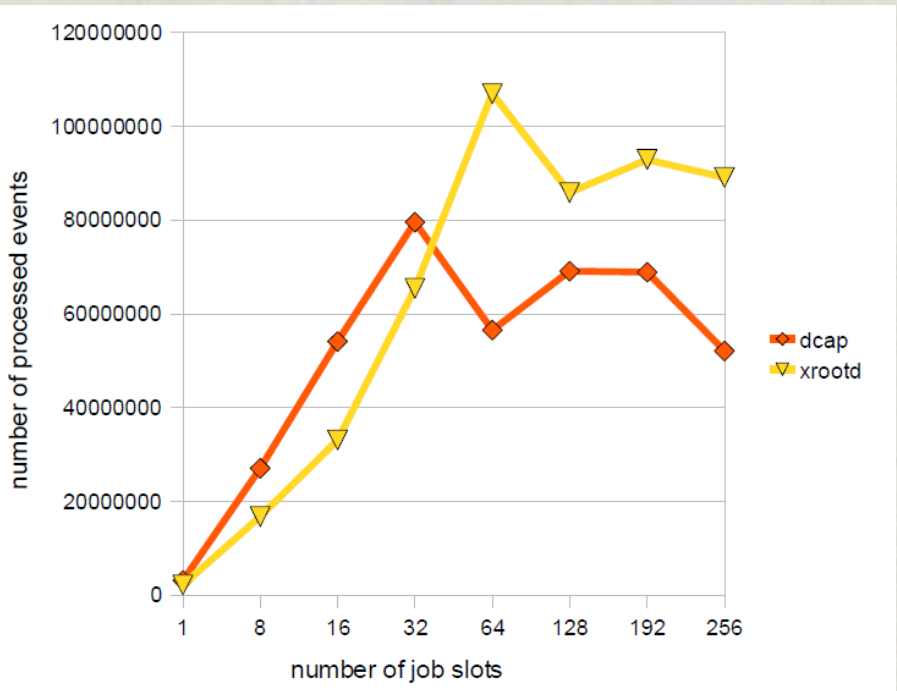


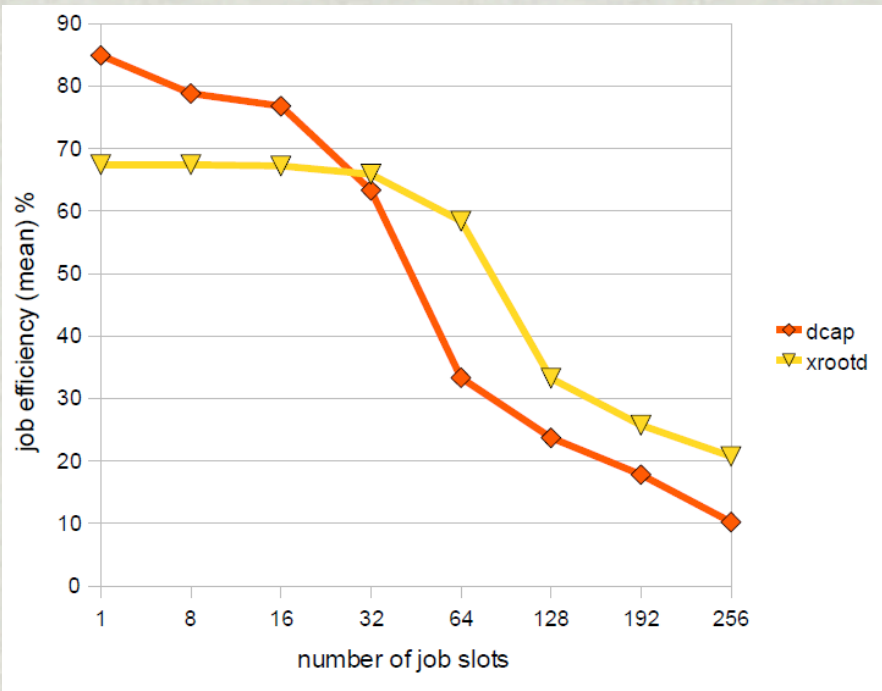
Source: HEPiX working group, Referenced by “Xrootd in dCache - design and experiences”, G.Behrmann, D.Ozerov and T.Zangerl, Contrib. to the CHEP 2010; in preparation.

Setup details: http://w3.hepix.org/storage/hep_pdf/2010/Spring/Maslennikov.SWG.Progress.Rep.pdf

Hammercloud tests for dcache 1.9.10

- XROOTD scales better

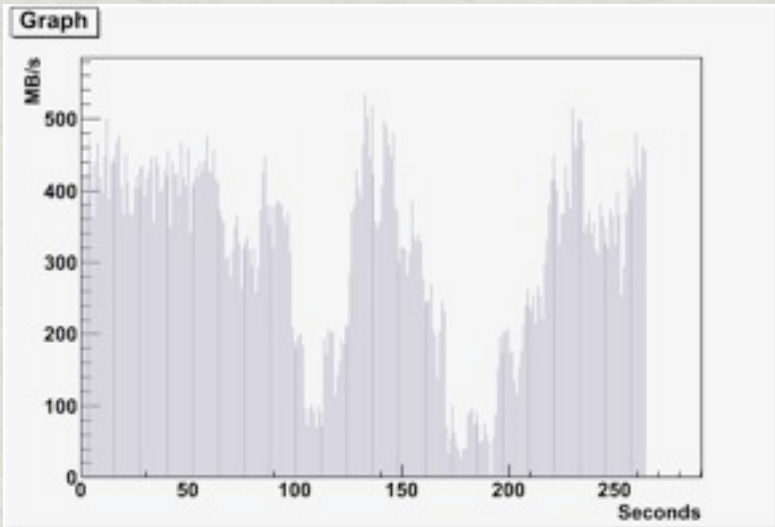




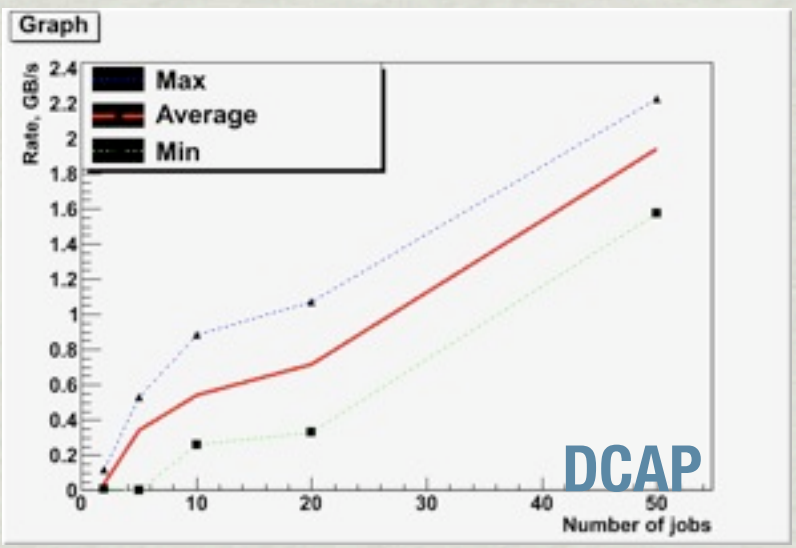
D. C. van der Ster, J. Elmsheuser, M. U. Garcia, and M. Paladin, “HammerCloud: A stress testing system for distributed analysis,” in Computing in High Engergy and Nuclear Physics - CHEP 2010, 2011.

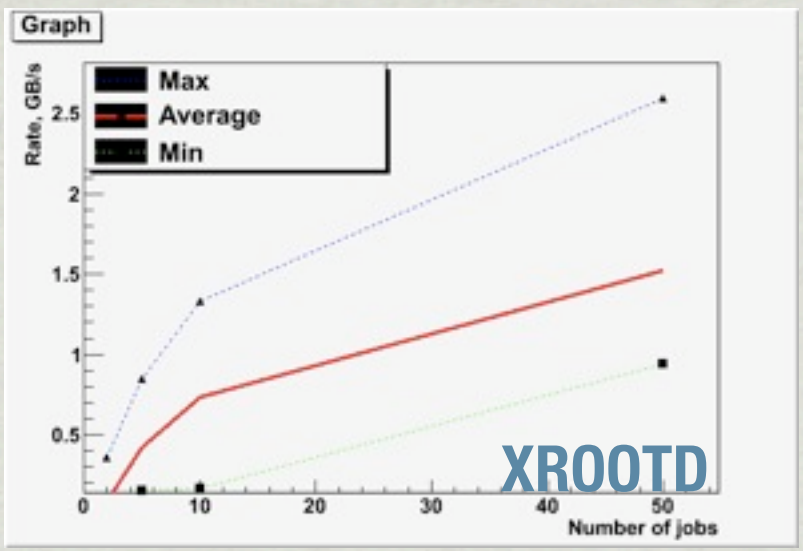
LoadTestEx for 1.9.5

A simple performance test which submits jobs into a queue. These dccp/xrdcp jobs are started synchronously and read files into memory. No ‘analysis’, just copying .



http://www.atlas.uni-wuppertal.de/~skalinin/LoadTestEx.tar





Considerations

- DCAP is sync; XROOTD and NFS4.1 can work asynchronously
- Client side caching
  - dcap only with the most recent version(dcap++) but not for vector read
  - NFS4.1 uses file system cache
  - Client caching in ROOTD xrootd driver

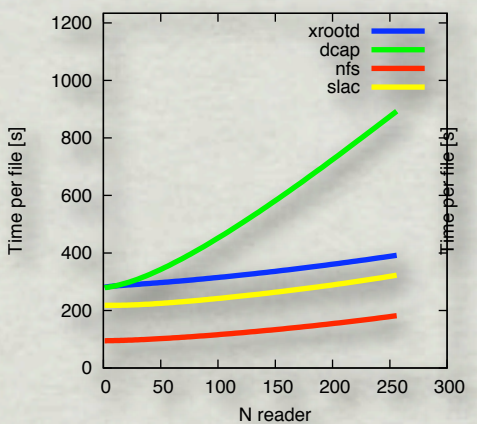
Conclusions

- dCache.org is continuously evaluating realistic applications(Hammercloud , ROOT) using different protocols.
- The performance difference between protocols highly depends on the access profile and the application.
- dCache/xrootd and XRootD/xrootd have comparable performances starting with 1.9.10
- Starting with 1.9.11 dCache/xrootd provides GSI authentication
- NFS4.1 shows very promising results and mostly better than other protocols
- Sites are encouraged to migrate to the next golden release of dCache

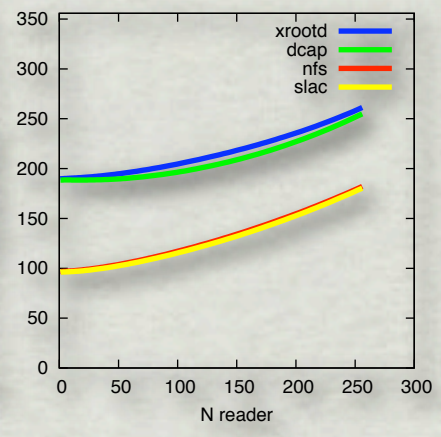
Backup slides

Results for optimized files

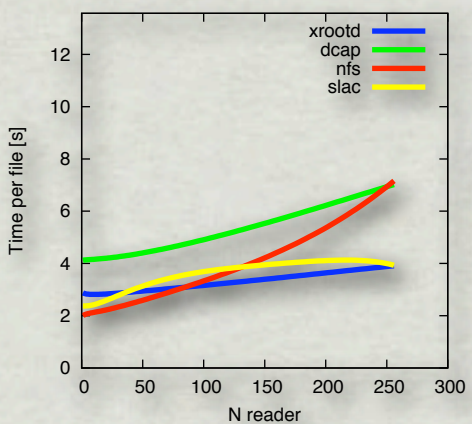
0B TreeCache all branches



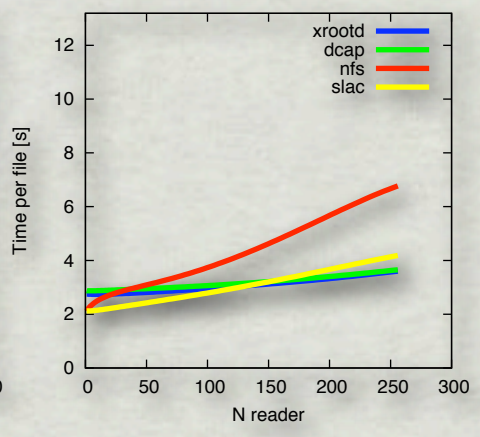
60MB TreeCache all branches



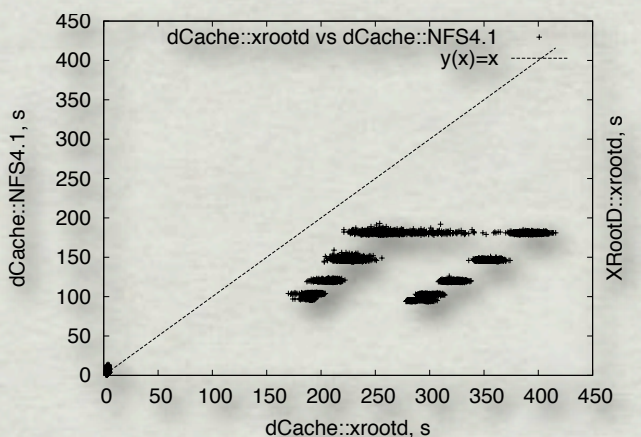
0B TreeCache 2 branches

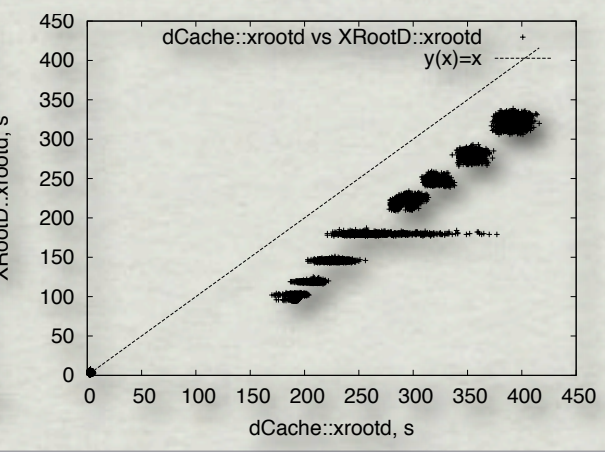


60MB TreeCache 2 branches



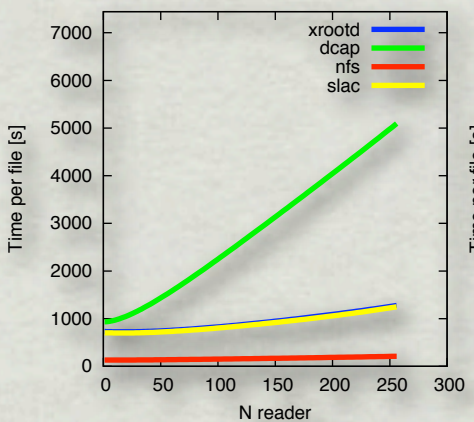
The results above can be combined into scatter plots. The same files, the same analysis script, the same worker nodes. The only differences were either protocols(NFS or xrootd) or server(dCache or XRootD)



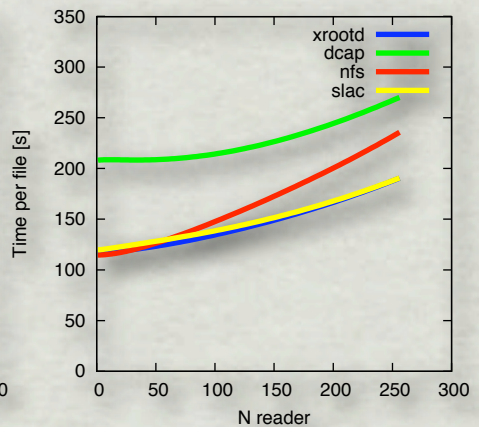


Results for not optimized files

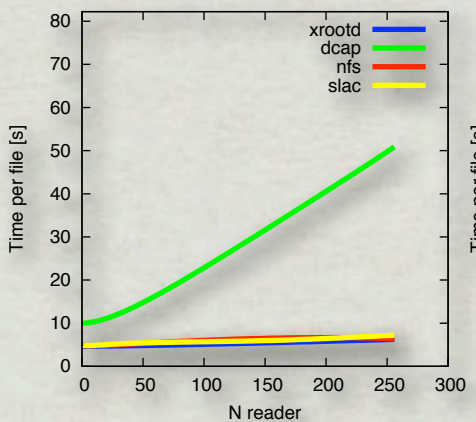
0B TreeCache all branches



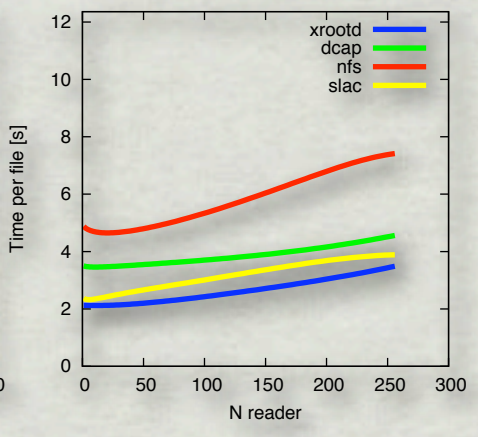
60B TreeCache all branches



0B TreeCache 2 branches



60B TreeCache 2 branches



The results above can be combined into scatter plots. The same files, the same analysis script, the same worker nodes. The only differences were either protocols(NFS or xrootd) or server(dCache or XRootD)