XRootd, disk-based, caching-proxy for optimization of data-access, data-placement and data-replication

L.A.T.Bauerdick[1], K.Bloom[3], B.P.Bockelman[3], D.C.Bradley[4], S.Dasu[4], J.M.Dost[2], I.Sfiligoi[2], A.Tadel[2], M.Tadel[2], F.Wuerthwein[2], A.Yagil[2]

for the **AAA project** and the **CMS collaboration**

[1] FNAL, [2] UC San Diego, [3] University of Nebraska-Lincoln, [4] University of Wisconsin-Madison

Introduction: Following the great success of AAA and FAX data federations of all US ATLAS & CMS T1 and T2 sites, AAA embarked on exploration of extensions of the federation architecture by developing two sample implementations of an XRootd, disk-based, caching-proxy:

1. ***Prefetching of complete files initiated by the file open request.***
   Suitable for cases when random file-access is expected or when it is known ahead of time that a significant fraction of files will be read.
2. ***On-demand caching of partial files, using relatively large block sizes.***
   Main use-case: improve robustness and provide automatic healing of HDFS based storage when additional replicas exist in the federation.

Both proxies are currently undergoing preproduction testing at UCSD.

Acknowledgments:

This work was supported by the following grants:

- **AAA:** NSF grants PHY-1104549, 1104447 & 1104664
- **CMS M&O:** NSF grant PHY-0612805
- **OSG:** DOE grant DE-FC02-06ER41436 sub. 647F290 and NSF grant PHY-1148698

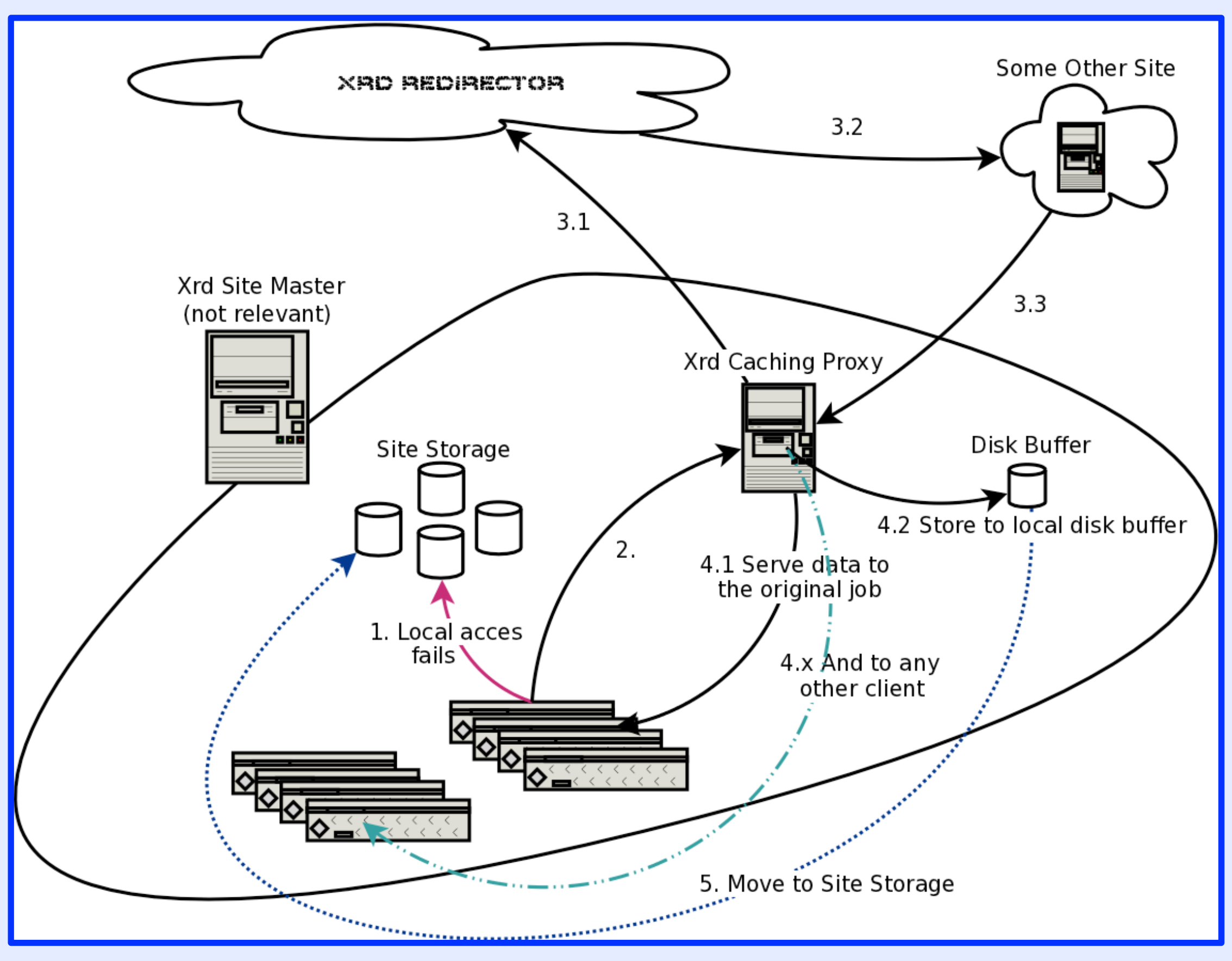
References & Pointers:

- **AAA project** at CHEP-2012
- **XRootd** *http://xrootd.org/*
- **GLED** *http://gled.org/* monitoring
- K.Bloom, *CMS Use of a Data Federation*, at this CHEP

1. Complete-file auto-prefetching proxy

Optimize access to remote data for random access & reuse.

1. On file-open local cache is checked for existence and state of the requested file.
2. If complete file is not available, prefetching of the file is started immediately, using configurable block size (1 MB). With few exceptions, blocks are retrieved in file-order:
   - First and last part of the file are retrieved first.
   - If client asks for data that is not yet available, required blocks get put to the beginning of the prefetch queue.
3. The prefetcher maintains a bit-field of downloaded blocks (saved every N blocks) to recover in case of a forced restart.
4. The info file also contains information about file accesses: open & close time, # of bytes read and # of requests.
5. Based on access information various cache reclamation strategies can be implemented.



2. Partial-file block-based on-demand proxy

Use remote data to heal and to supplement local storage.

1. Nothing is done on file-open.
2. Only when a request comes proxy checks if the block exists on disk and, if it doesn't, prefetches the block in whole. Block-size is passed as URL parameter (e.g. 64 or 128 MB).
3. Request is served as soon as the data is available.
4. Each block is stored as a separate file, post-fixed by block size and beginning offset – facilitates reinjection into HDFS.

3. HDFS extension for using XRootd fallback

Allow HDFS storage to run seamlessly with lower replication.

1. Internally, HDFS client attempts to retrieve and verify check-sums of all block replicas before throwing an *IOException*.
2. Our specialization of *DFSInputStream* catches those and:
   - instantiates an *XrdBlockFetcher* object (on first error only);
   - uses it to retrieve the requested data via XRootd;
   - remembers bad blocks to avoid further delays.
3. Of course, *XrdBlockFetcher* is pointed at the partial-file block-based on-demand proxy implementation.
4. Retrieved blocks can be re-injected into HDFS to replace lost blocks or to provide additional replicas under heavy load.

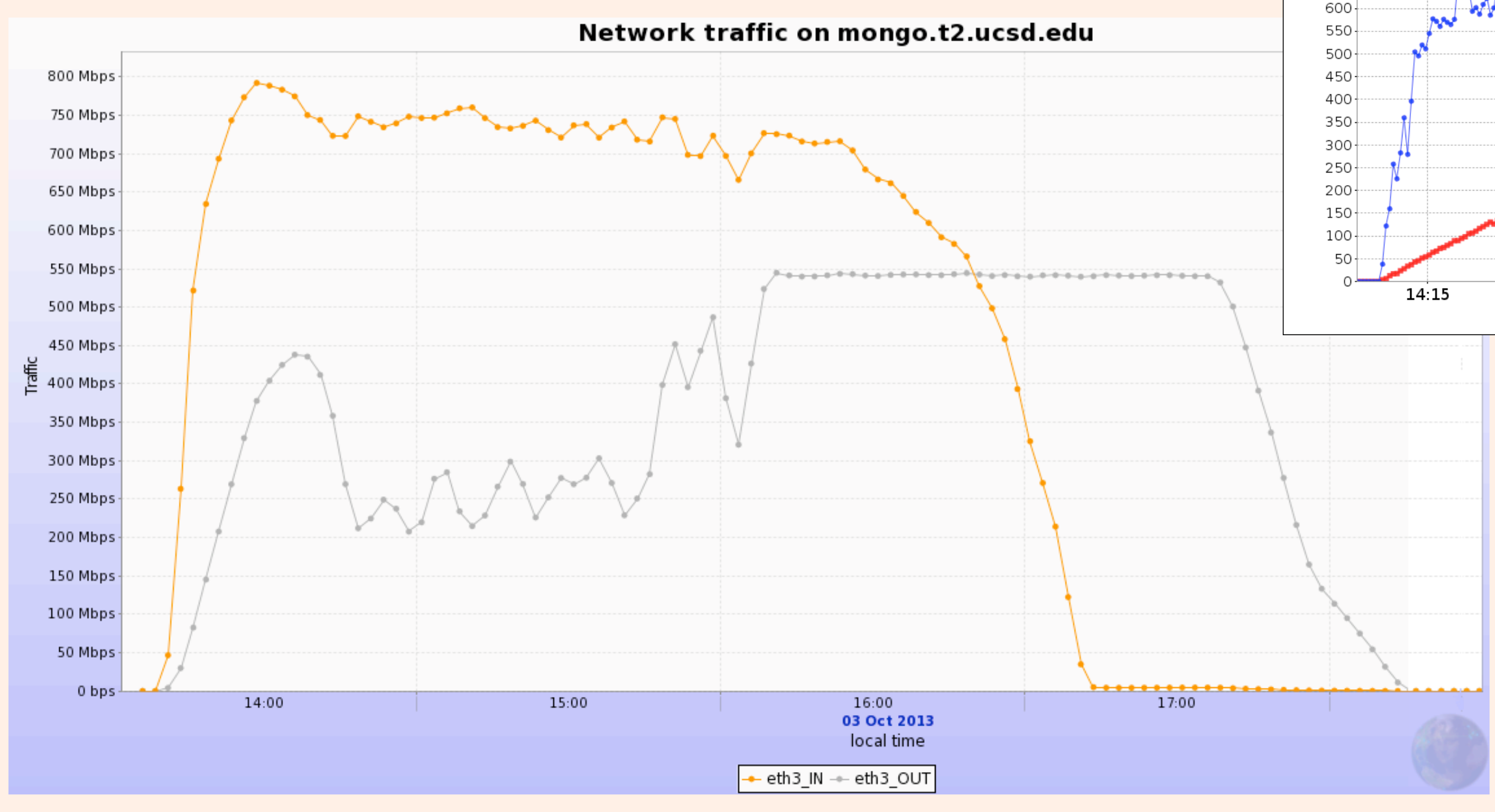
Required minimal changes to HDFS code:

- change some DFSInputStream members protected;
- allow instantiation of a different InputStream class based on file URI;
- biggest problem is packaging and deployment – using OSG repository.

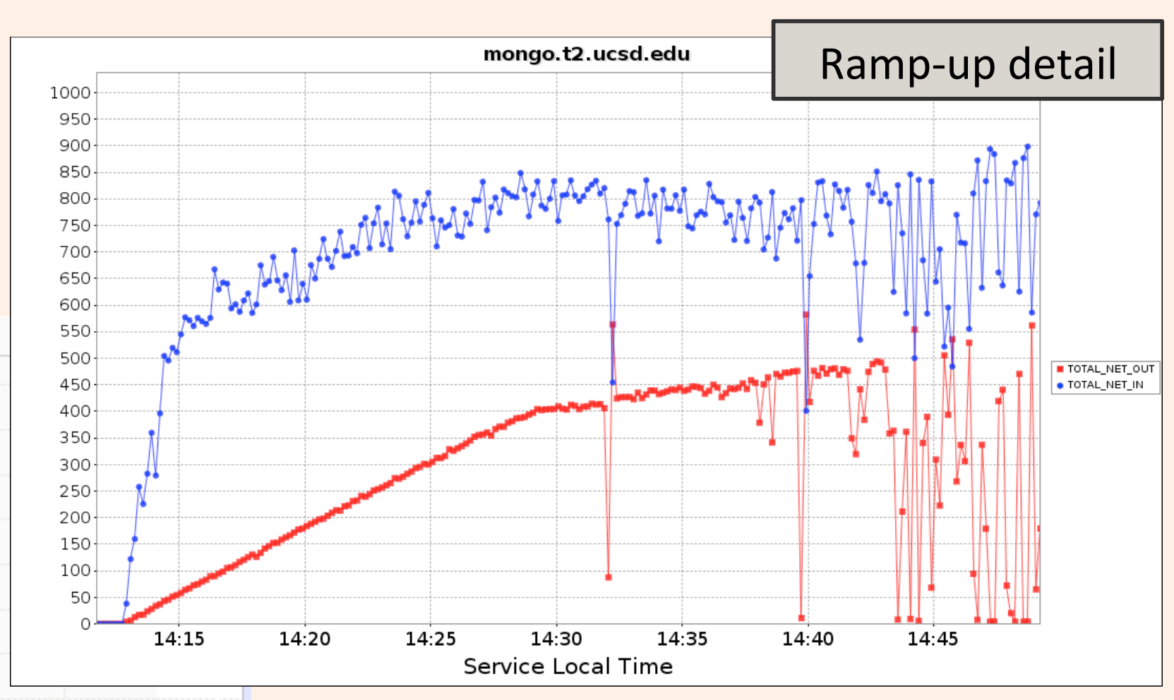
XrdBlockFetcher:

- all XrdClient handling implemented in C++ using JNI;
- reporting via UDP to allow monitoring of operation & performance.

Caching-proxy scaling test



Ramp-up detail



This problem seems network-stack related, there were no other indications of machine being stressed.

A 5-year old machine had trouble with both disk I/O and network interrupts, causing lower input traffic (600Mbps) and earlier saturation (about 140 jobs).

The machine: 12-core Xeon E5-2620, 64GB RAM, ZFS with RAID-5. One 1 Gbps NIC was used in the test.

The test, designed to cause trouble:

1. Start with empty cache, files are all in US
2. Run 250 jobs on several machines, each requiring 2.4 MB every 10 s (240kB/s avg).
3. Slow ramp-up: start jobs 5 seconds apart
4. Sit back and see where things break …

Conclusion: One standard machine can serve about 200 average jobs. *To be continued …*

Analysis of detailed monitoring data from *Jun 21, 2012* to *Aug 31, 2013* presented encyclopedically for estimation of caching-proxy operational parameters

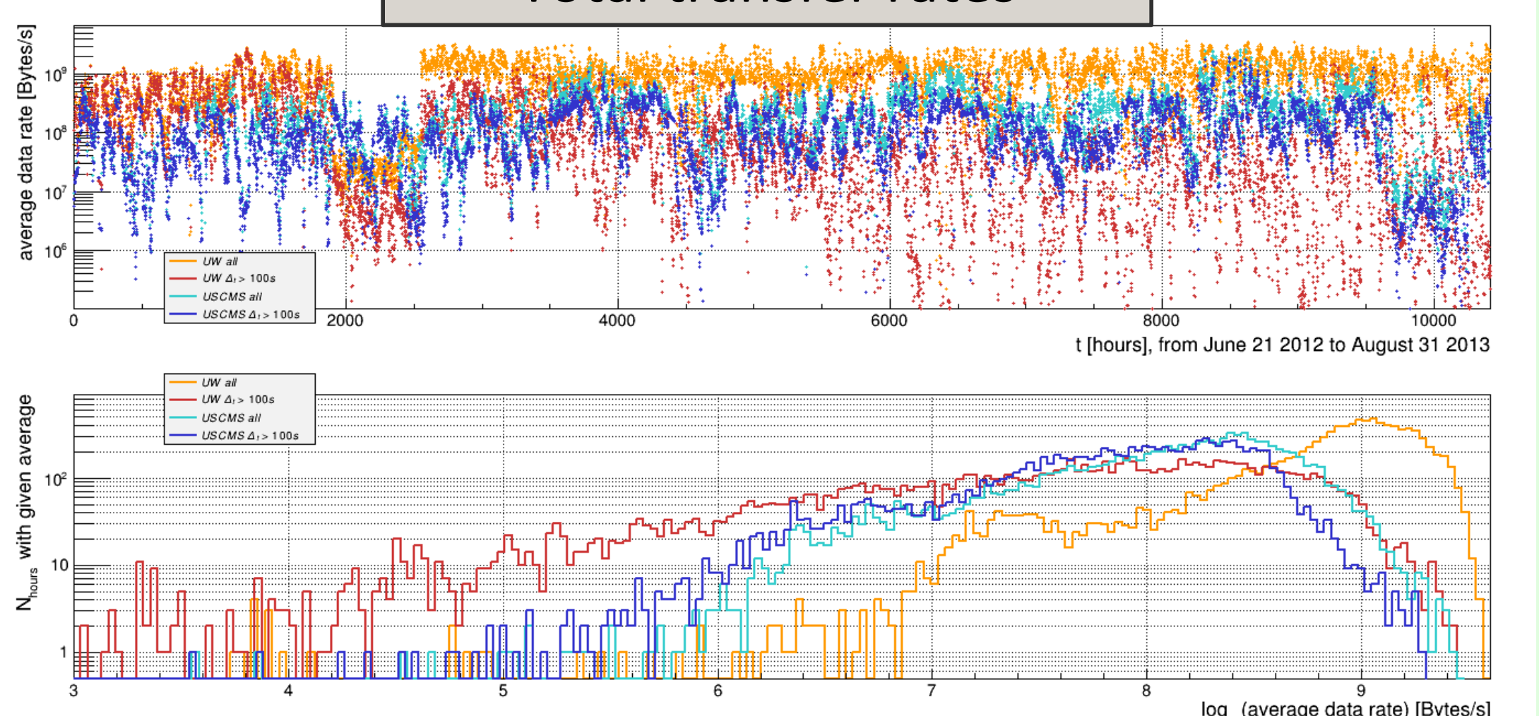
Data from AAA XRootd detailed monitoring was used. Two main sub-sets were analyzed:

1. Local access at U. of Wisconsin-Madison
   UW uses XRootd for internal data access, too!
2. Remote access among all US CMS sites

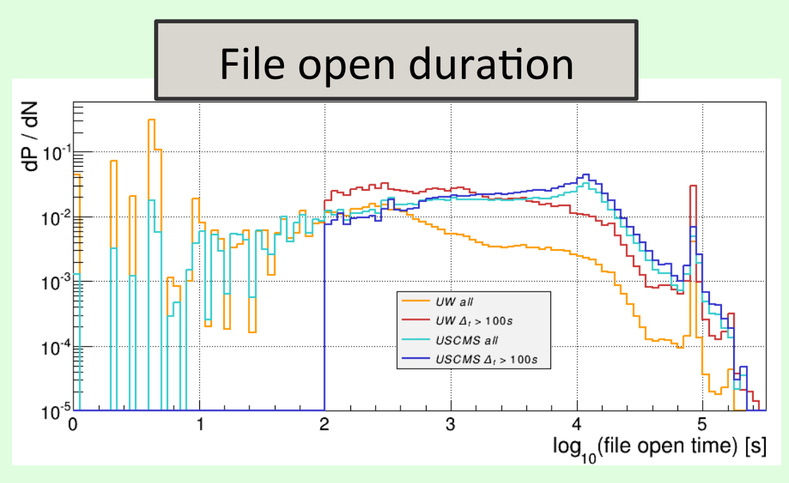
Selection	UW	US-CMS
all with preload	56,528,278	6,310,605
t > 100 s, no preload	7,839,134	4,534,612
t < 100 s, no preload	35,672,599	703,793

Storage area	UW	US-CMS
PhEDEx	4,173,396	3,310,554
User & group	3,665,698	724,004

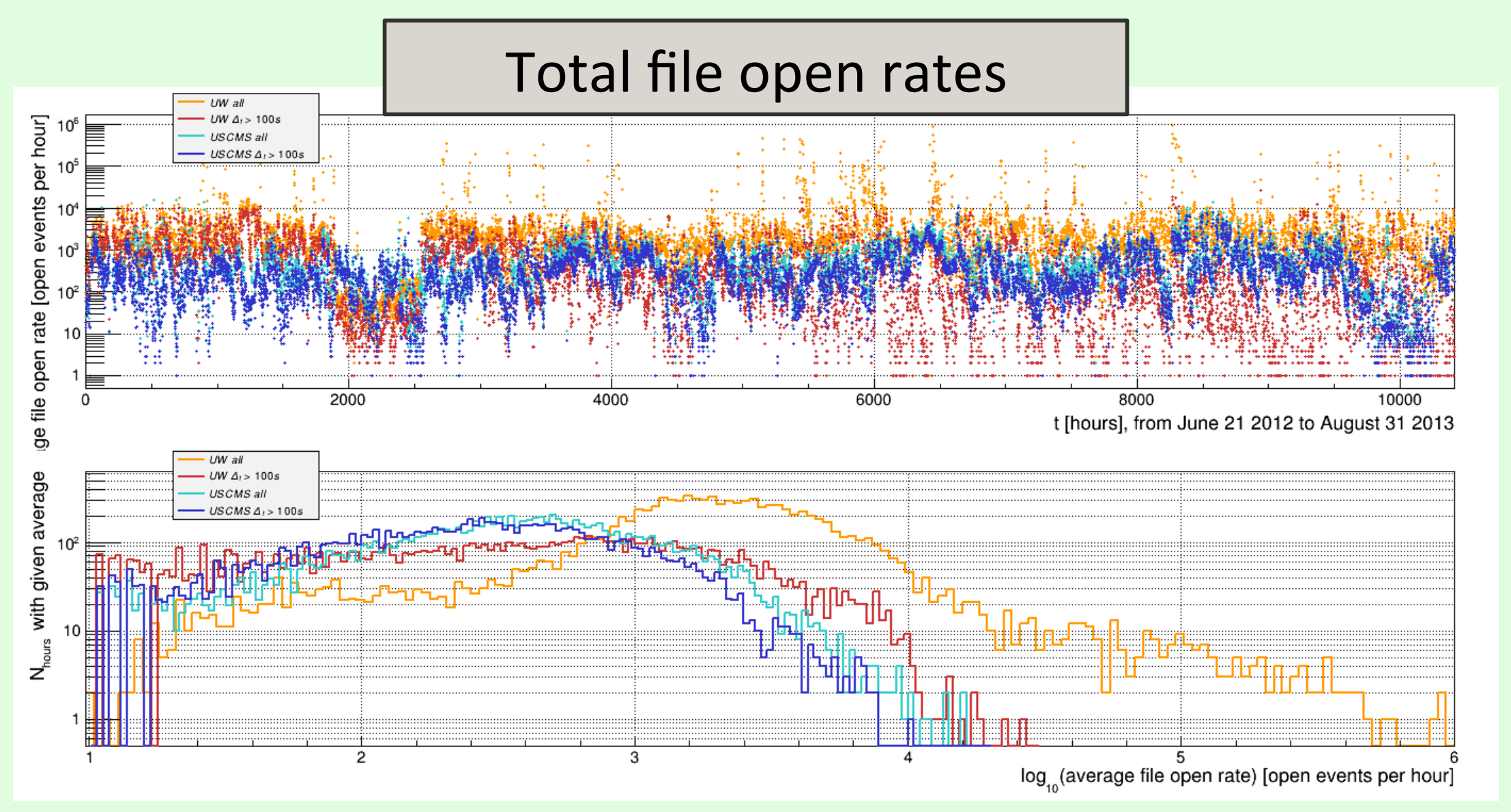
Total transfer rates



File open duration

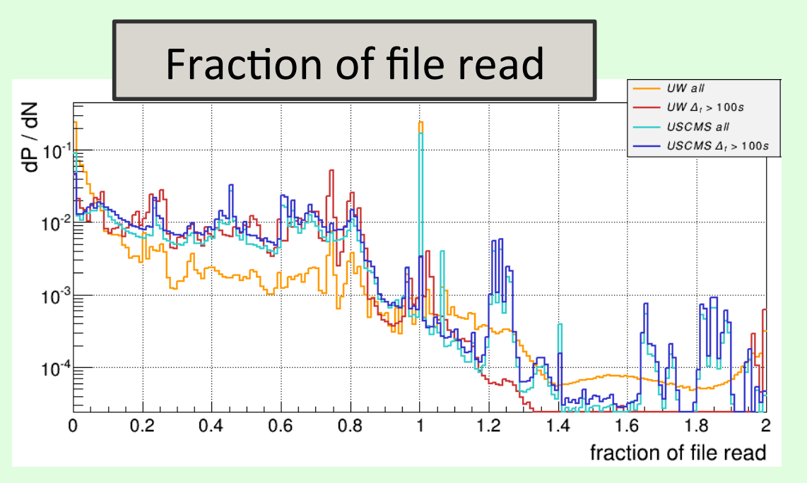


Total file open rates

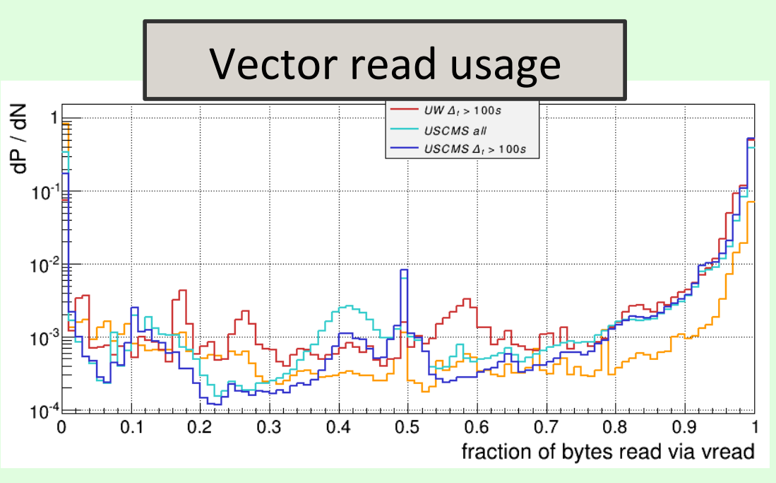


Note the log-log plots:
Observed ranges are huge!
It took us about 2 weeks to understand *most* details here …

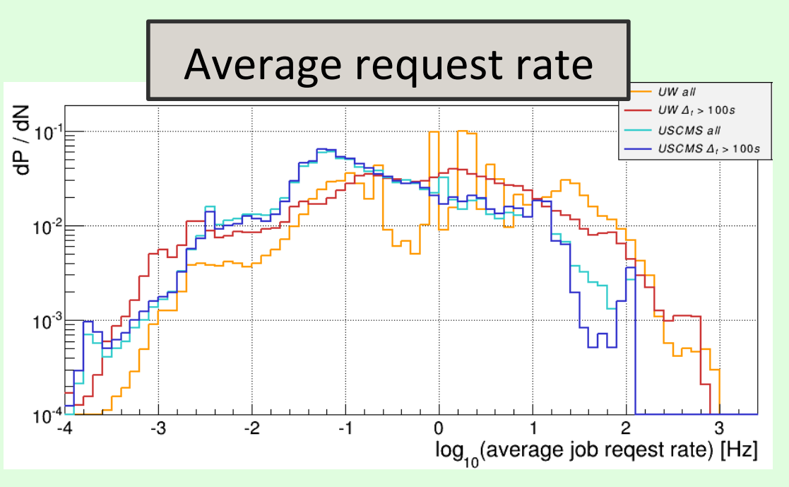
Fraction of file read



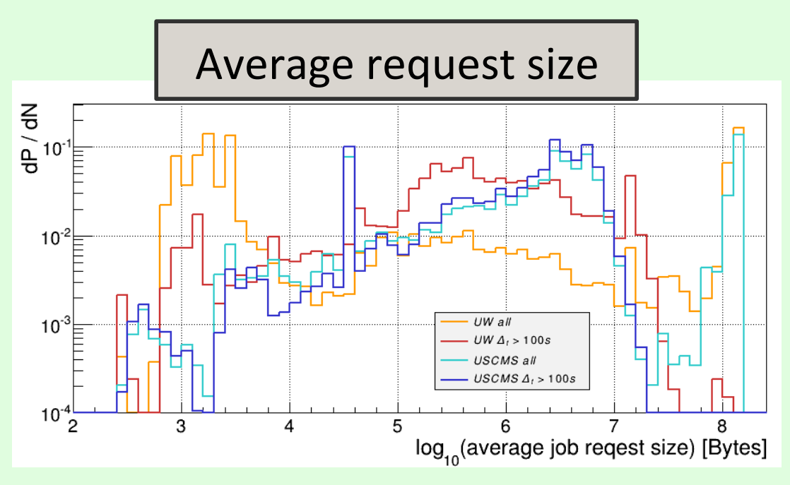
Vector read usage



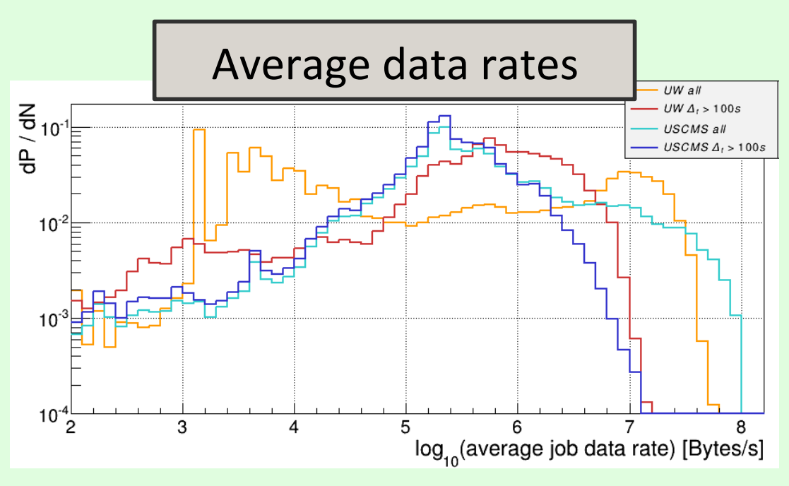
Average request rate



Average request size



Average data rates



Further work:

- Integration of caching with job scheduling.
- Interaction of cache with local storage.
- Study various deployment strategies within a T2.

Take home message: Monitor your proxy in detail & analyze its usage regularly. Start with well defined data-sets and job-types, be careful with user analysis ☺