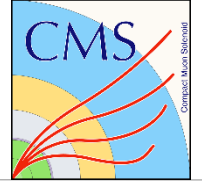


Caltech

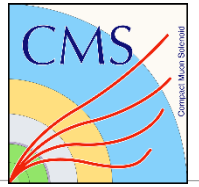
HTTP as a Data Access Protocol: Trials with XrootD in CMS' AAA Project

J.R. Vlimant, J. Balcas on behalf of the entire team

The 22nd International Conference on Computing in High Energy and Nuclear Physics, CHEP 2016

San Francisco, 13/10/2016

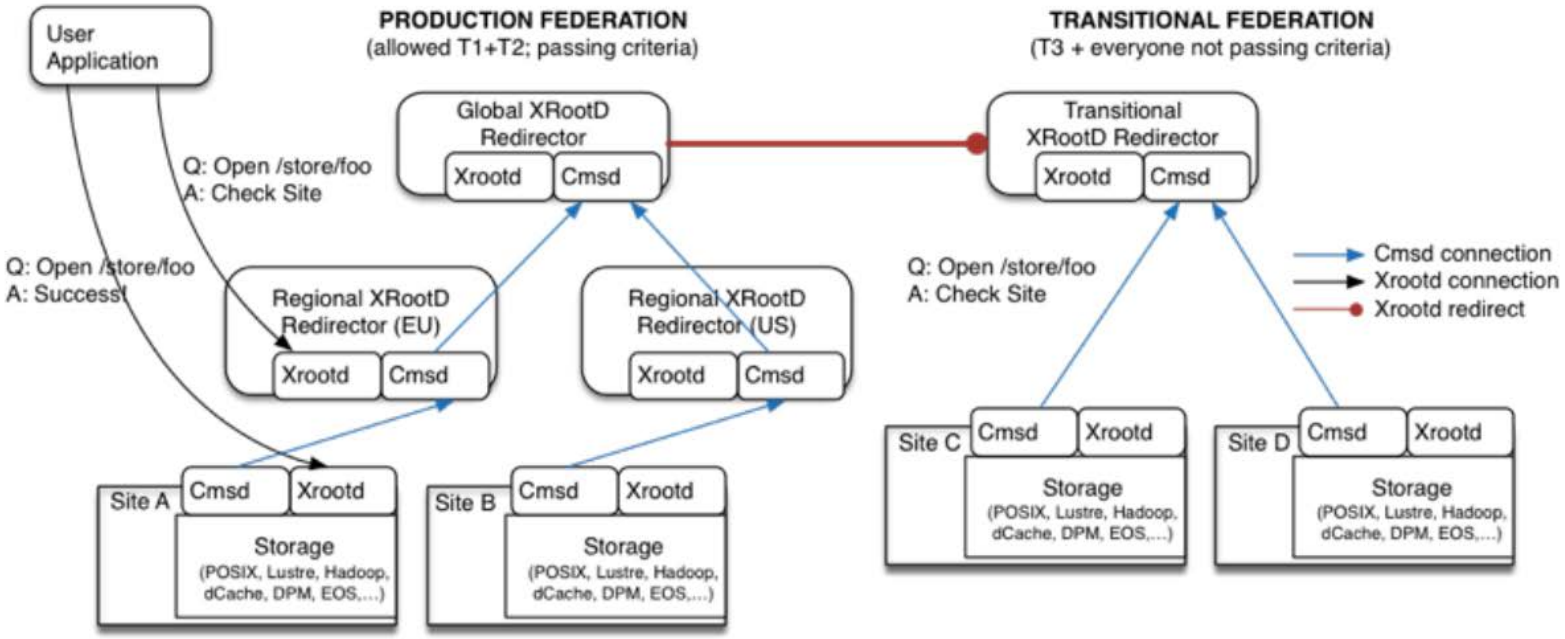
Motivations and Goals

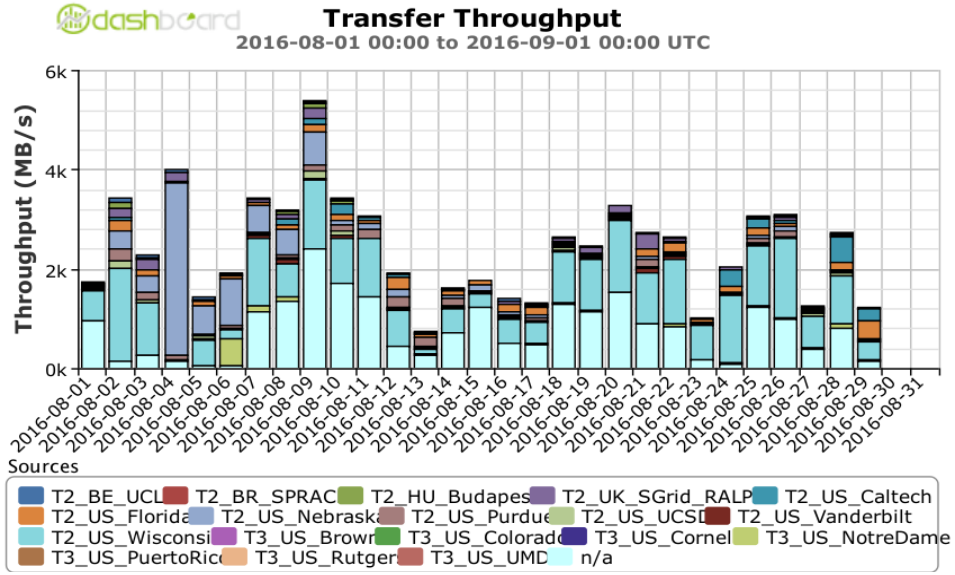


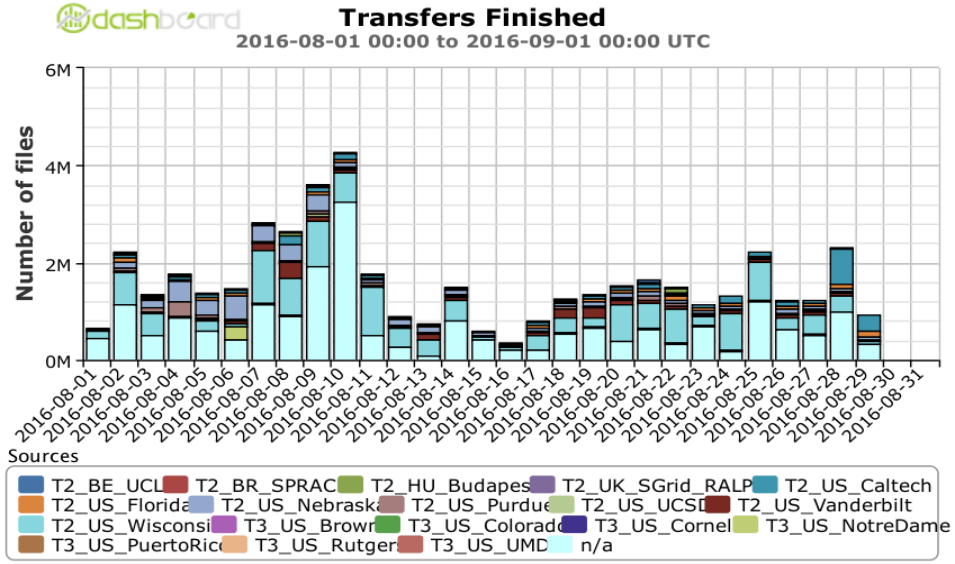
Caltech

- Motivations:
  - CMS Workflow processing requires remote reads (Pileup mixing)
  - XrootD is not supported everywhere (Opportunistic)
  - Not all sites support XRootD
- Goals:
  - Demonstrate ability of using HTTP data federations **in a manner analogous to today's AAA infrastructure**.
  - Improve CMSSW's HTTP support **in case we see increased use of HTTP as a transport protocol.**
  - Extended to S3 and WebDav.

Any Data, Any Time, Anywhere







AAA data federation based on XRootD, this system provides uniform remote data access to all of CMS's on-disk data.

Workflow management software: HTCondor and glideinWMS provides computing resource provisioning.

Global file systems: To distribute CMS software, we utilize CernVM File System (CVMFS) and have integrated it with Parrot to emulate it on hosts where it is otherwise unavailable.

HTTP Data Federation Project in CMS (Operational)

✓ Deliverables: Operational

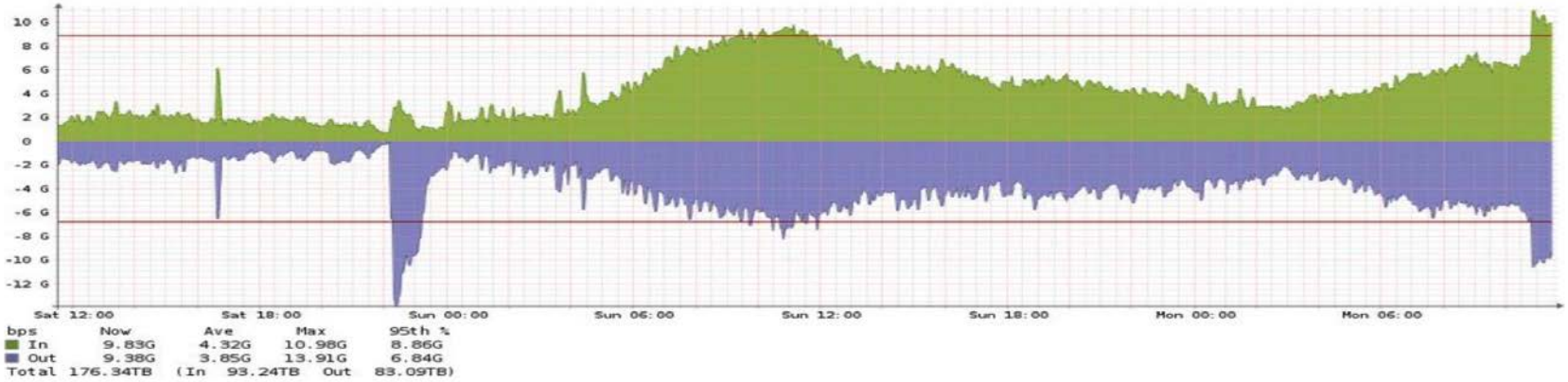
A. Improve support infrastructure at Caltech for data federations; Aim for a system that can saturate 80 Gbps

B. Enable HTTP support at the US redirector and at the Caltech site. In collaboration with AAA and Caltech T2

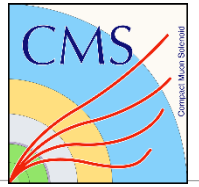
A. New Infrastructure:

- Started from 1 XRootD server at 10Gbps and moved to 8 active xrootd servers total: **4 production + 4 testbed**
- Set up production testbed, added an SSD to one of the nodes to test local FS performance
- New **100Gbps switch between CACR and Lauritsen.** Was commissioned on July 2016
- Moving IPAC servers to Lauritsen => heavy data replication took place and link from CACR to Lauritsen was upgraded to 40GE. **No apparent congestions or large spikes were noticed.**

B. HTTP support is enabled in Caltech and Nebraska



HTTP Data Federation Project in CMS (Software)



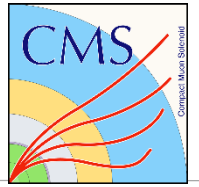
Caltech

✓ Software Deliverables

- Implement a libdavix-based StorageFactory plugin in CMSSW to replace the current fork/exec of curl for HTTP access
- Develop extensions to the Xrootd [HTTP implementation] to replace any identified missing functionality (e.g. client-based monitoring IDs)
- As necessary, optimize the HTTP implementation within Xrootd to improve any performance deficiencies observed in the tests below
- As necessary, develop patches to better integrate HTTP-over-Xrootd with the OSG distribution (e.g. Authorization)
- (OPTIONAL) Develop asynchronous APIs to libdavix analogous to the current XrootD ones (so we can develop a multistream client for HTTP)

- **Plugin is available in CMSSW release**
  - Supports HTTP(s) protocols (http, https)
  - Allows to control plugin through environment variables (Davix_DEBUG, X509_*, /tmp/x509up_u%d)
  - Vector operations (Partial reads, multi-range, single range) and get full file content to I/O buffer.

HTTP Data Federation Project in CMS (Knowledge)



Caltech

❑ Deliverables: Knowledge Base

- ➢ Document configuration & deployment of HTTP-over-Xrootd at Caltech
- ➢ Measure operational performance of a single Xrootd daemon at exporting w/HTTP
- ➢ Measure comparative performance in using the Caltech T2 via standard Xrootd versus HTTP through the AAA redirector
- ➢ Measure the comparative performance in using 'curl'-based StorageFactory plugin versus the libdavix one above
- ➢ Characterize the performance of using Xrootd for HTTP caching versus squid

- **Working on understanding Hadoop performance on US CMS Sites:**
  - Caltech (dd - 650Mbps, xrdcp – 350Mbps), Wisconsin (xrdcp – 350Mbps), Nebraska (asked to do same tests). All tests are done locally from HDFS to /dev/null.
- **Measured performance of CMSSW caching algorithms and different protocol providers (**Apache, Nginx, HttpOverXrootd**)**
- **Constructing testbed for HDFS and CEPH storage to estimate the interplay with CMS workflows.**

Testbed Setup (Based at Caltech)

❑ Hardware & Software:

- ❑ Data Node: CC7, 24gb RAM, 60TB RAID 0, Intel(R) Xeon(R) CPU X5670@2.93Ghz
- ❑ Execute Node: SLC6, 64GB RAM, 4TB RAID 0, Intel (R) Xeon(r) CPU E5-2670@2.60Ghz
- ❑ No linux or software modifications, default installation. Increased **NIC MTU** and **txqueuelen**

❑ Test cases:

- ❑ Always running only one job on execute node which reads from Data node on a specific protocol.

Service/Plugin	Davix			XRootD			Curl		
Apache	Appl	Storage	LazyD						LazyD
Nginx	Appl	Storage	LazyD						LazyD
XrootD	Appl	Storage	LazyD	Appl	Storage	LazyD			LazyD

Percent of Branches/ Trigger Factor [1]	0	1	20
1			
30			
100			

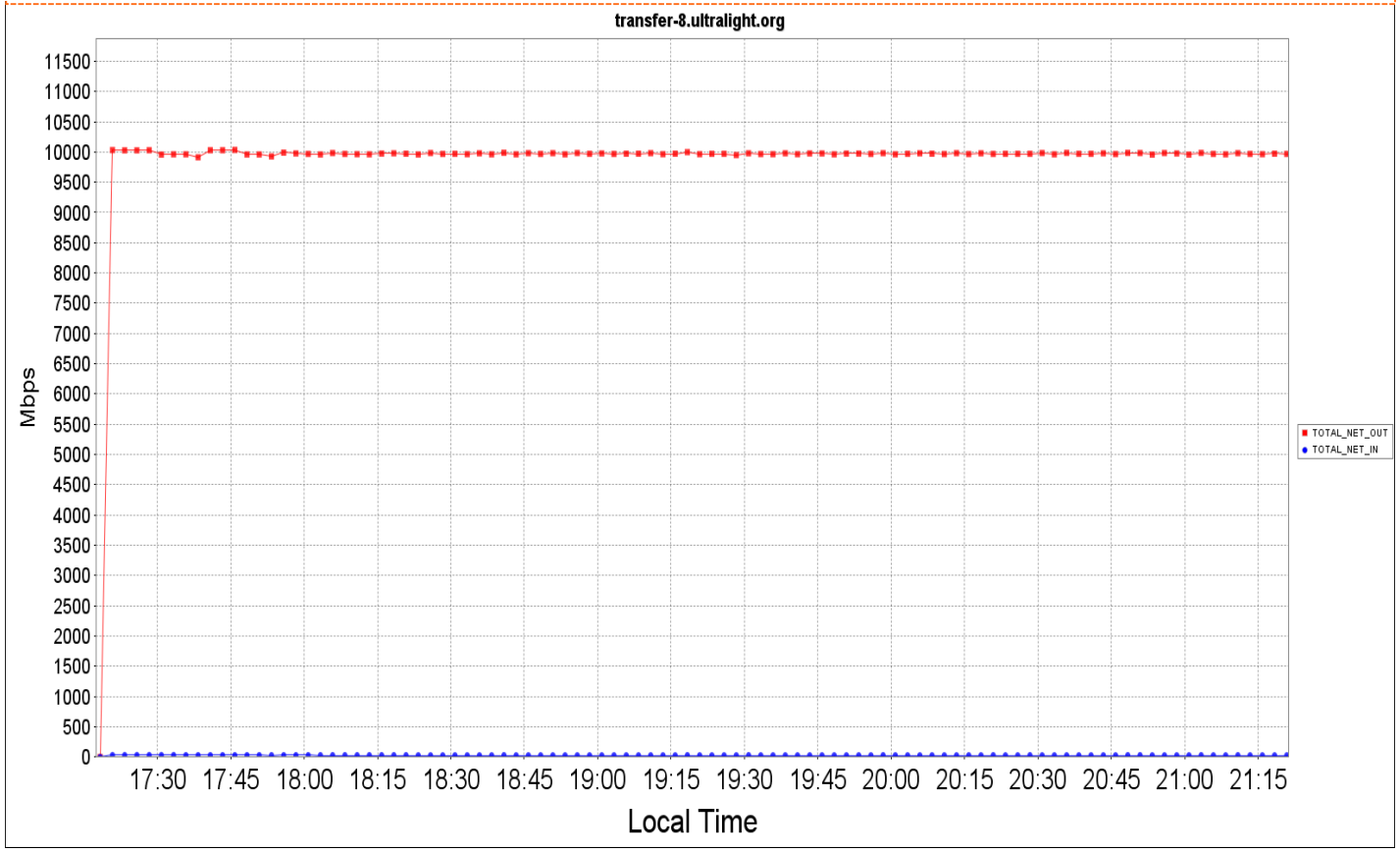
JobN/Delay	0	2	5	10	15	20	25	30	40	50	60	70	80	90	100
0															
1															
2															
3															
4															
5															
6															
7															
8															
9															

[1] - https://github.com/cms-sw/cmssw/blob/CMSSW_8_1_X/IOPool/Input/doc/IOExerciser-README

Tesbed environment performance

Caltech

Nettest – stable 9.9 Gb/s



Used tools:
FDT – Application For Fast Data Transfers over WAN
Monalisa – Monitoring agents for the end hosts

Storage write/read speed

- Data node
  - Timing cached reads:
    18312 MB in 2 seconds = **9.2 GB/sec**
  - Timing buffered disk reads:
    6974 MB in 3 seconds = **2.3 GB/sec**
  - Write speed:
    2.1 GB copied, 1.70 s, **1.3 GB/s**
- Execute node
  - Timing cached reads:
    17884 MB in 2 seconds = **9.0 GB/sec**
  - Timing buffered disk reads:
    5952 MB in 3 seconds = **1.9 GB/sec**
  - Write speed:
    2.1 GB copied, 2.2 s, **1.0 GB/s**

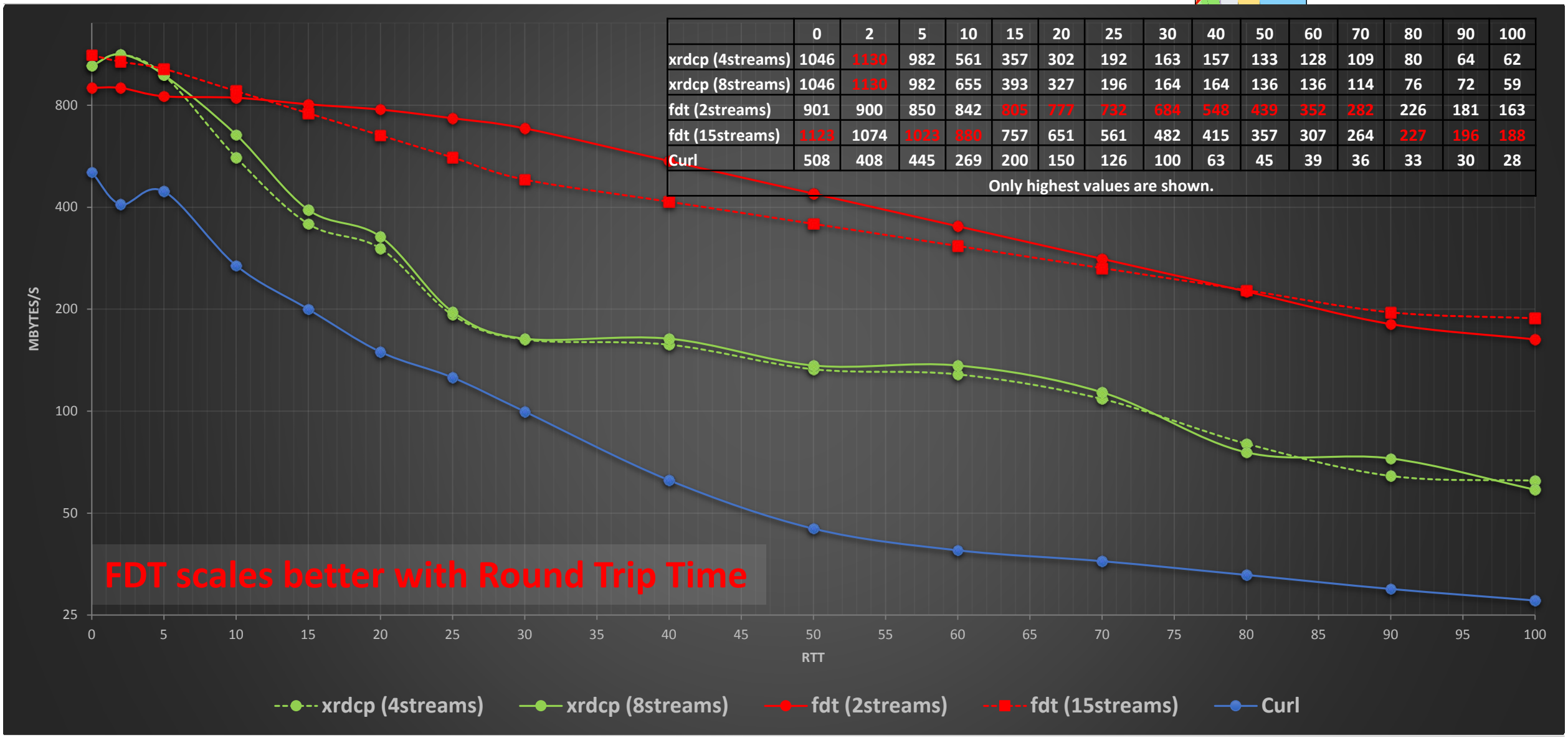
✓ **No limitations from hardware**

Transfer test: xrdcp vs curl vs fdt (Mbytes/s)

Caltech

	0	2	5	10	15	20	25	30	40	50	60	70	80	90	100
xrdcp (4streams)	1046	1130	982	561	357	302	192	163	157	133	128	109	80	64	62
xrdcp (8streams)	1046	1130	982	655	393	327	196	164	164	136	136	114	76	72	59
fdt (2streams)	901	900	850	842	805	777	732	684	548	439	352	282	226	181	163
fdt (15streams)	1123	1074	1023	880	757	651	561	482	415	357	307	264	227	196	188
Curl	508	408	445	269	200	150	126	100	63	45	39	36	33	30	28

Only highest values are shown.

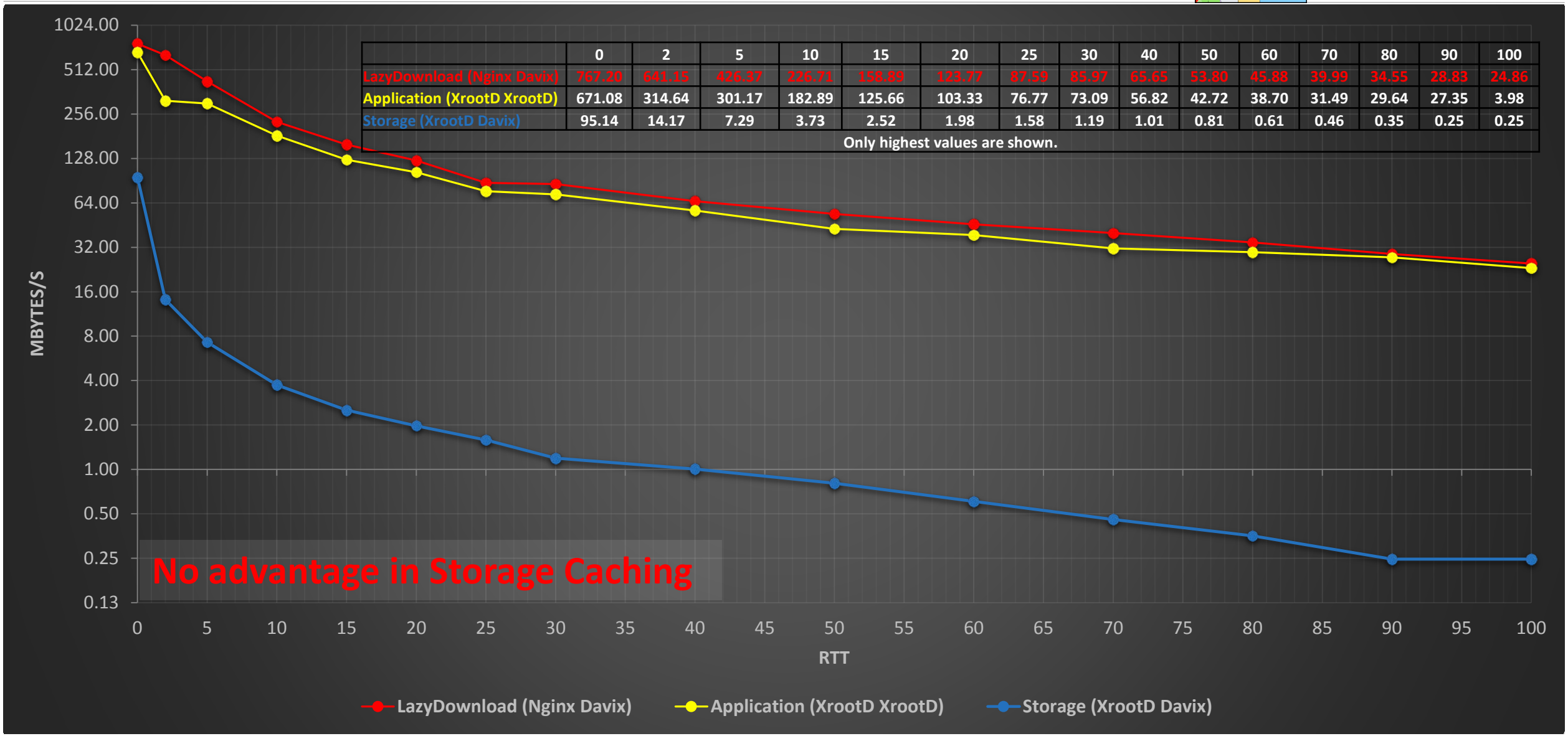


Compare CMSSW caching types (Mbytes/s)

Caltech

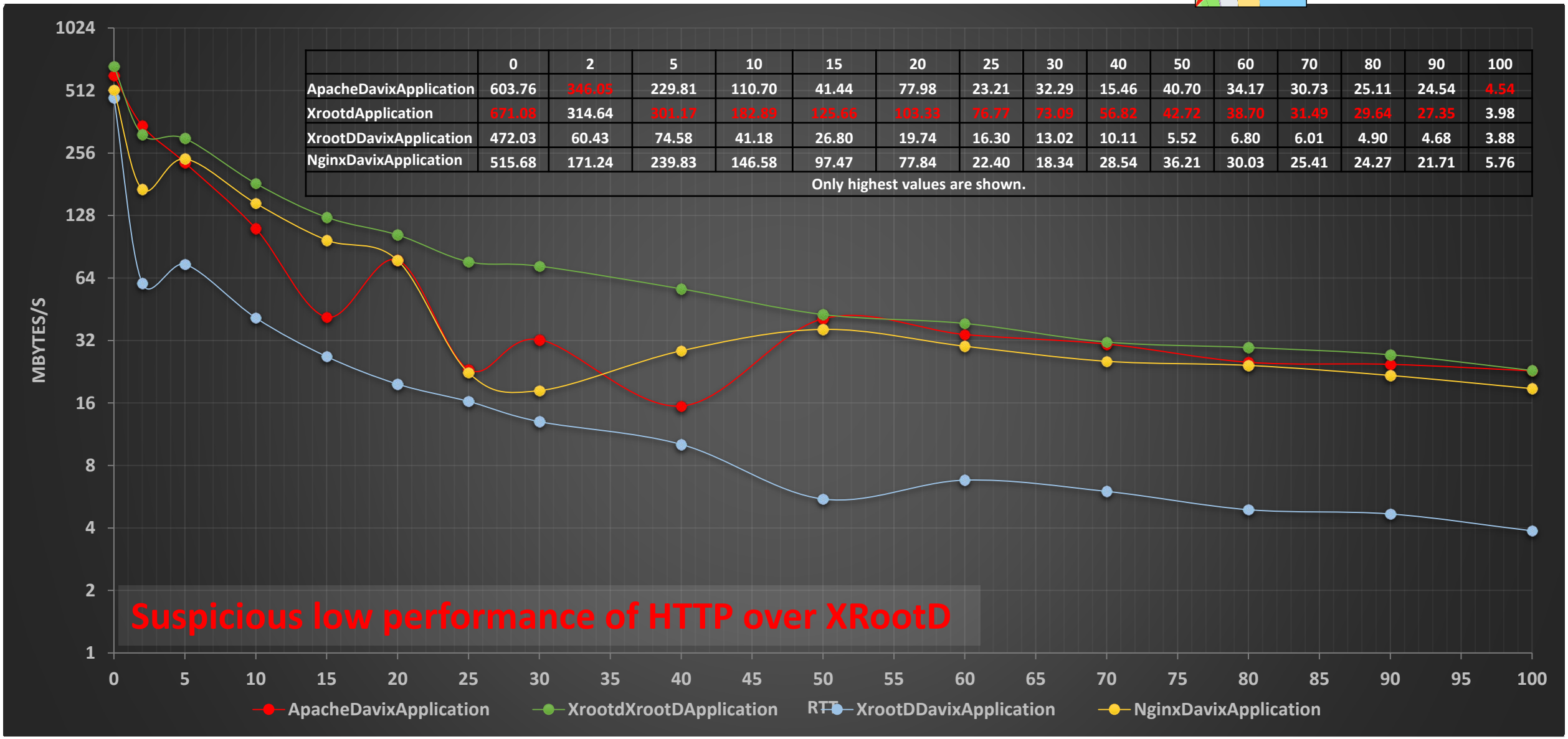
	0	2	5	10	15	20	25	30	40	50	60	70	80	90	100
LazyDownload (Nginx Davix)	767.20	641.15	426.37	226.71	158.89	123.77	87.59	85.97	65.65	53.80	45.88	39.99	34.55	28.83	24.86
Application (XrootD XrootD)	671.08	314.64	301.17	182.89	125.66	103.33	76.77	73.09	56.82	42.72	38.70	31.49	29.64	27.35	3.98
Storage (XrootD Davix)	95.14	14.17	7.29	3.73	2.52	1.98	1.58	1.19	1.01	0.81	0.61	0.46	0.35	0.25	0.25

Only highest values are shown.



Application caching (Mbytes/s)

Caltech



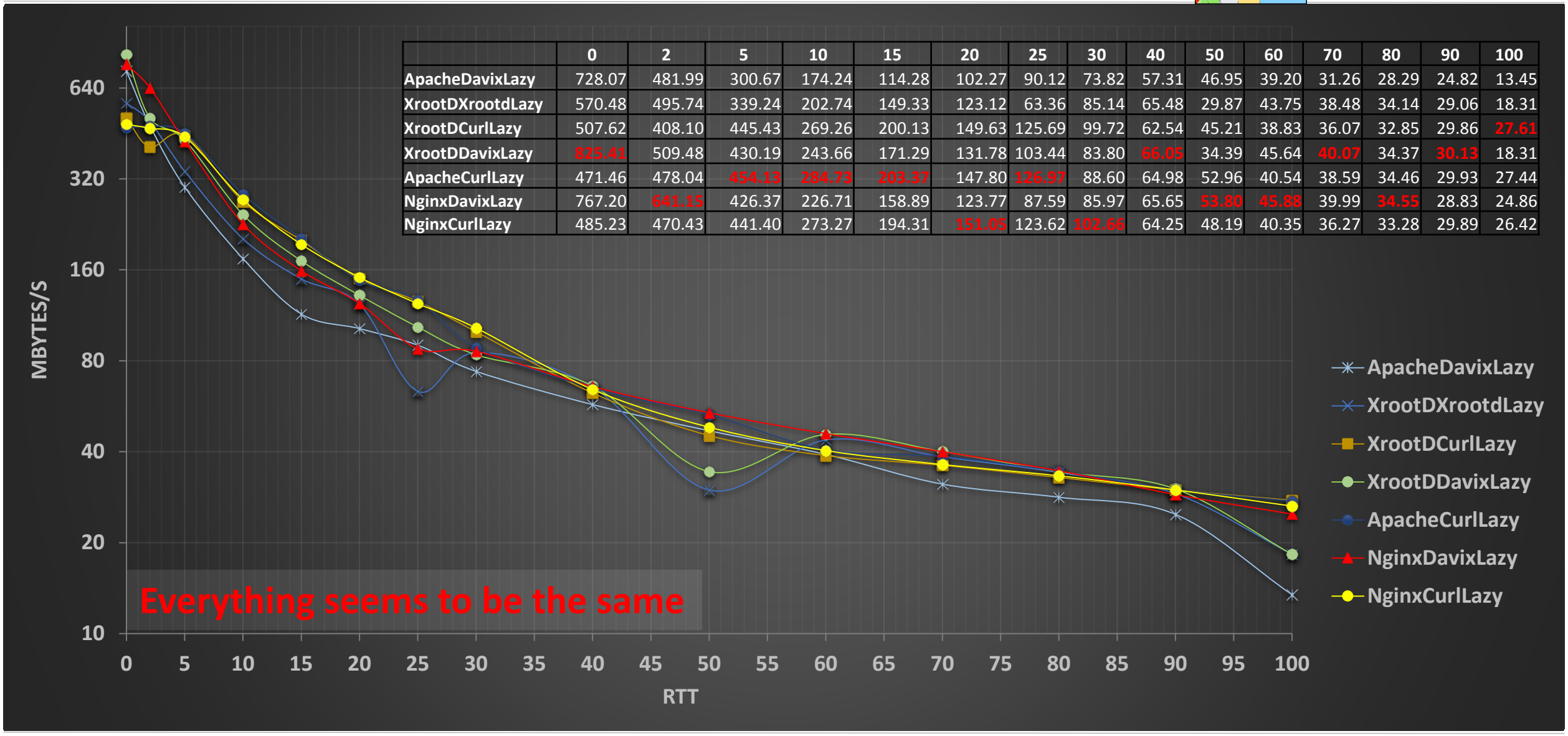
	0	2	5	10	15	20	25	30	40	50	60	70	80	90	100
ApacheDavixApplication	603.76	346.05	229.81	110.70	41.44	77.98	23.21	32.29	15.46	40.70	34.17	30.73	25.11	24.54	4.54
XrootdApplication	671.08	314.64	301.17	182.89	125.66	103.33	76.77	73.09	56.82	42.72	38.70	31.49	29.64	27.35	3.98
XrootDDavixApplication	472.03	60.43	74.58	41.18	26.80	19.74	16.30	13.02	10.11	5.52	6.80	6.01	4.90	4.68	3.88
NginxDavixApplication	515.68	171.24	239.83	146.58	97.47	77.84	22.40	18.34	28.54	36.21	30.03	25.41	24.27	21.71	5.76

Only highest values are shown.

Lazy download caching (Mbytes/s)

Caltech

	0	2	5	10	15	20	25	30	40	50	60	70	80	90	100
ApacheDavixLazy	728.07	481.99	300.67	174.24	114.28	102.27	90.12	73.82	57.31	46.95	39.20	31.26	28.29	24.82	13.45
XrootDXrootdLazy	570.48	495.74	339.24	202.74	149.33	123.12	63.36	85.14	65.48	29.87	43.75	38.48	34.14	29.06	18.31
XrootDCurlLazy	507.62	408.10	445.43	269.26	200.13	149.63	125.69	99.72	62.54	45.21	38.83	36.07	32.85	29.86	27.61
XrootDDavixLazy	825.41	509.48	430.19	243.66	171.29	131.78	103.44	83.80	66.05	34.39	45.64	40.07	34.37	30.13	18.31
ApacheCurlLazy	471.46	478.04	454.13	284.73	203.37	147.80	126.97	88.60	64.98	52.96	40.54	38.59	34.46	29.93	27.44
NginxDavixLazy	767.20	641.15	426.37	226.71	158.89	123.77	87.59	85.97	65.65	53.80	45.88	39.99	34.55	28.83	24.86
NginxCurlLazy	485.23	470.43	441.40	273.27	194.31	151.05	123.62	102.66	64.25	48.19	40.35	36.27	33.28	29.89	26.42



Summary

- Testbed setup at Caltech, installed and used for performance tests
- Implemented Davix Storage Factory plugin in CMSSW
  - Performed initial performance test with XRootD, Davix, Curl
- CMSSW caching outcome:
  - Storage caching is slowest. Huge impact on total job efficiency through any access type starting from 2ms delay.
  - HTTP Application caching shows much better performance through Apache or Nginx than through HTTP over XrootD
  - Lazy download - All are near same bounds with average small difference. Davix plugin through Nginx or HTTP over XrootD always shows a bit better throughput than XrootD plugin

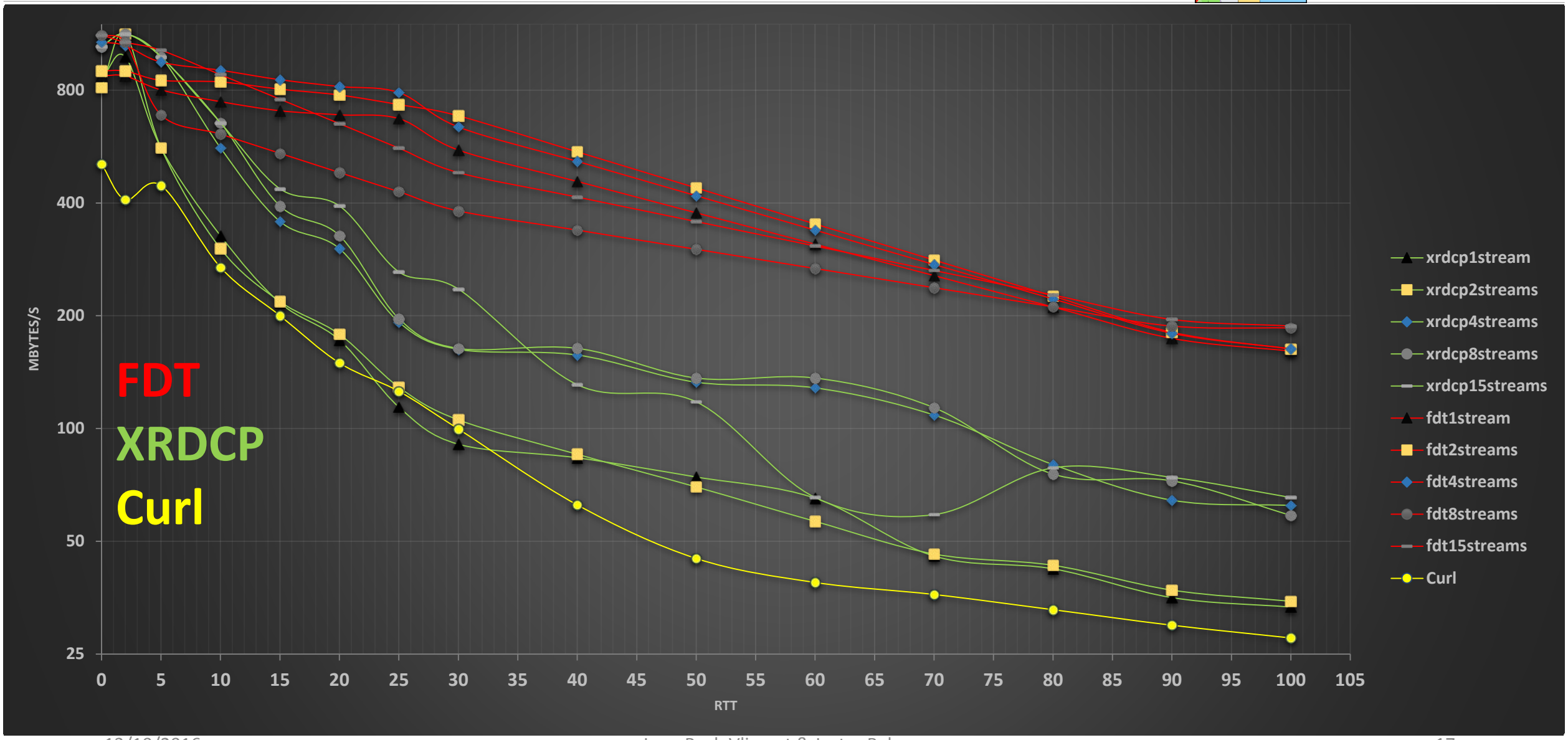
Outlook

- Implement S3 and WebDav in CMSSW and have it in next CMSSW release
- Average production job Runtime is ~5 hours and LONG TCP connections lead to failures/inefficiency's:
  - Once TCP connection made, in case of failure, whole job is considered as failed, no retries to re-open a file;
  - Network, IO, CPU usage changes over the time;
- Develop retry logic in CMSSW and allow sites control it. Also jobs could overwrite it through configuration knobs:
  - How many times to retry
  - Timeout between retries
- Move all xrootd servers to CC7 (Issues with SLC6)
- Improve XrootD monitoring to see the correct throughput, num. of connections, failures.
- Make sure failed sites are reported
- Collect and analyze production job efficiency metrics in real time and improve their efficiency

BACKUP SLIDES

Comparing xrdcp vs curl vs fdt (Mbytes/s)

Caltech

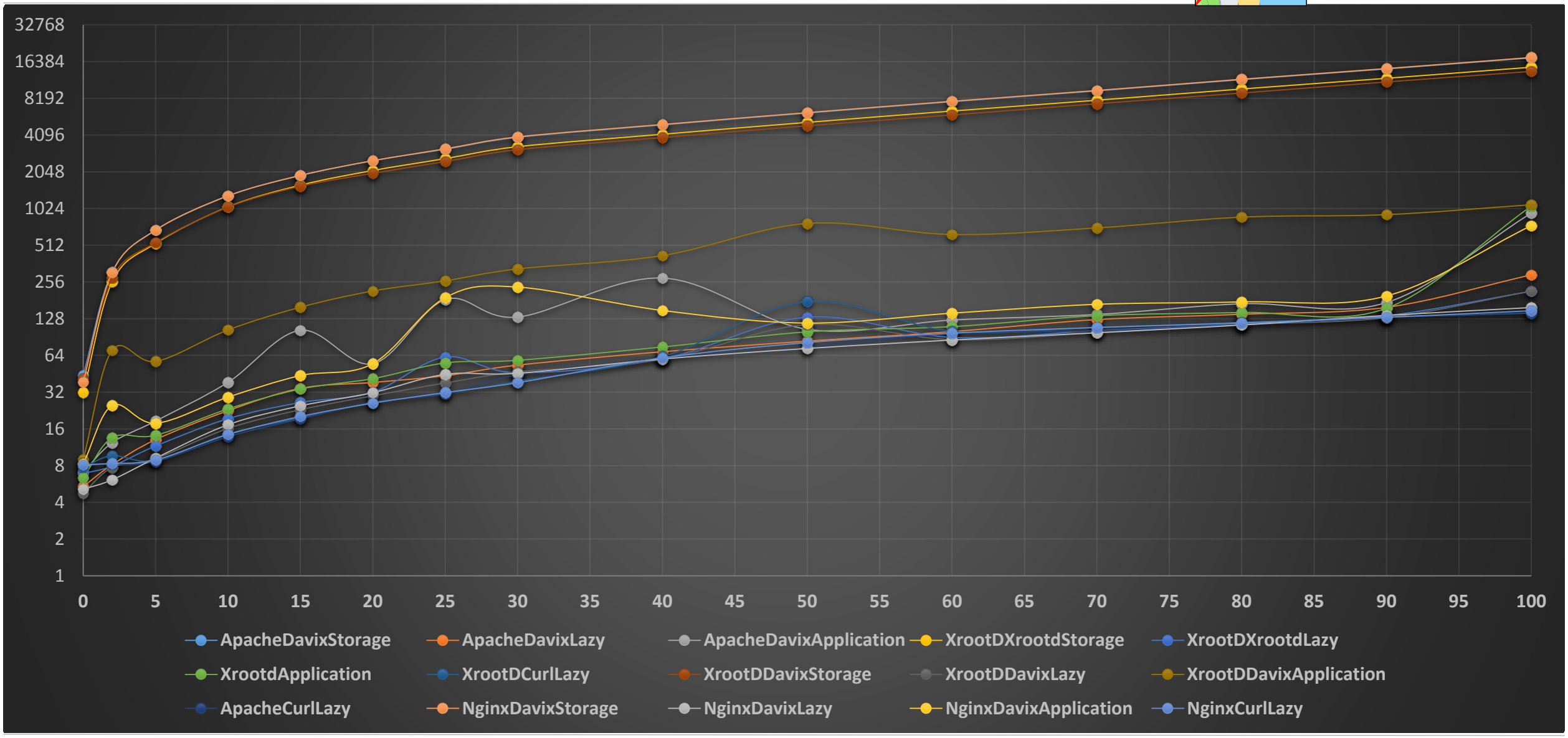


Storage caching (Mbytes/s)

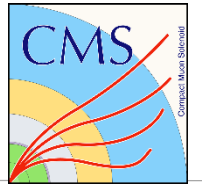
Caltech

	0	2	5	10	15	20	25	30	40	50	60	70	80	90	100
ApacheDavixStorage	89.43	12.70	5.76	3.02	2.06	1.56	1.26	0.99	0.84	0.66	0.50	0.42	0.34	0.21	0.13
XrootDXrootdStorage	123.04	15.31	7.42	3.72	2.47	1.87	1.50	1.12	0.95	0.76	0.57	0.43	0.34	0.24	0.13
XrootDDavixStorage	95.14	14.17	7.29	3.73	2.52	1.98	1.58	1.19	1.01	0.81	0.61	0.46	0.35	0.25	0.25
NginxDavixStorage	101.62	12.82	5.77	3.03	2.05	1.56	1.25	0.98	0.83	0.66	0.50	0.42	0.34	0.21	0.12

Overall Results (Average read time in seconds)



CMSSW caching hints



Caltech

CMSSW normally reads data via a set of I/O adaptors. The interface between CMSSW and ROOT is called TFileAdaptor. The underlying C++ interface for POSIX-like storage objects is called StorageFactory.

- Application:
  - This is the default and means ROOT will do the caching. If PoolSource.cacheSize is non-zero, a TTreeCache of that size will be created per open file. Asynchronous read-ahead will be turned off and the cache will be filled with normal reads.
- Storage:
  - ROOT will drive the caching using a prefetch list, but will not allocate a cache of its own. ROOT will hand over the prefetch list to the storage layer, which is expected to do its own caching.
- Lazy-download:
  - Remote files will be downloaded to a local shadow file on demand in 128MB segments. ROOT reads will be directed to this local file; ROOT will never read directly from the remote file.

Software used in tests

- Apache is the most widely used web server software. Developed and maintained by Apache Software Foundation, Apache is an open source software available for free.
- Nginx is a web server. It can act as a reverse proxy server for TCP, UDP, HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer and an HTTP cache.
- XRootD software framework is a fully generic suite for fast, low latency and scalable data access, which can serve natively any kind of data, organized as a hierarchical filesystem-like namespace, based on the concept of directory.
- FDT is an Application for Efficient Data Transfers which is capable of reading and writing at disk speed over wide area networks (with standard TCP). It is written in Java, runs an all major platforms and it is easy to use.