2017

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

J. Balcas

B. P. Bockelman

D. Kcira

H. Newman

J. Vlimant

See next page for additional authors

Authors
J. Balcas, B. P. Bockelman, D. Kcira, H. Newman, J. Vlimant, and T. W. Hendricks

Journal of Physics: Conference Series

PAPER • **OPEN ACCESS**

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

To cite this article: J Balcas *et al* 2017 *J. Phys.: Conf. Ser.* **898** 062042

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 062042 doi:10.1088/1742-6596/898/6/062042

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

J Balcas[1], B P Bockelman[2], D Kcira[1], H Newman[1], J Vlimant[1], T W Hendricks[1] for the CMS Collaboration

[1]California Institute of Technology, Pasadena, CA 91125, USA
[2]University of Nebraska-Lincoln, Lincoln, NE 68588, USA

E-mail: justas.balcas@cern.ch

Abstract. The main goal of the project to demonstrate the ability of using HTTP data federations in a manner analogous to the existing AAA infrastructure of the CMS experiment. An initial testbed at Caltech has been built and changes in the CMS software (CMSSW) are being implemented in order to improve HTTP support. The testbed consists of a set of machines at the Caltech Tier2 that improve the support infrastructure for data federations at CMS. As a first step, we are building systems that produce and ingest network data transfers up to 80 Gbps. In collaboration with AAA, HTTP support is enabled at the US redirector and the Caltech testbed. A plugin for CMSSW is being developed for HTTP access based on the DaviX software. It will replace the present fork/exec or curl for HTTP access. In addition, extensions to the XRootD HTTP implementation are being developed to add functionality to it, such as client-based monitoring identifiers. In the future, patches will be developed to better integrate HTTP-over-XRootD with the Open Science Grid (OSG) distribution. First results of the transfer tests using HTTP are presented in this paper together with details about the initial setup.

1. Introduction

The Worldwide LHC Computing Grid [1] (WLCG) is a global collaboration between hundreds of computer centers. They are geographically distributed over the world with various size of number of cores and storage space for data. The WLCG provides data transfer services for the four main virtual organizations (CMS [2], Atlas [3], LHCb [4], Alice [5]) using two technologies: File Transfer Service (FTS3) [6] and XRootD [7]. XRootD is a distributed, scalable system for low-latency file access. It is the primary wide-area network data-access framework for the CMS experiment at the Large Hadron Collider (LHC). The motivation to implement the HTTP data federation in the present AAA (Any Data, Any Time, Anywhere) infrastructure [8] is to improve CMSSW [9] HTTP support for an eventual increased use of it as a transfer protocol. Furthermore, support for XRootD is not universal. Outside CMS many providers support S3 [10] or WebDav [11].

In this paper we start by presenting current technologies and their implementation details within CMS. Then we continue with describing the implementation of AAA XRootD HTTP based data access in CMSSW. Detailed results from the XRootD HTTP implementation tests and extension to S3 and WebDav protocols are then presented. In addition, we give an overview of the currently supported caching algorithms in CMSSW and results from tests that use them.

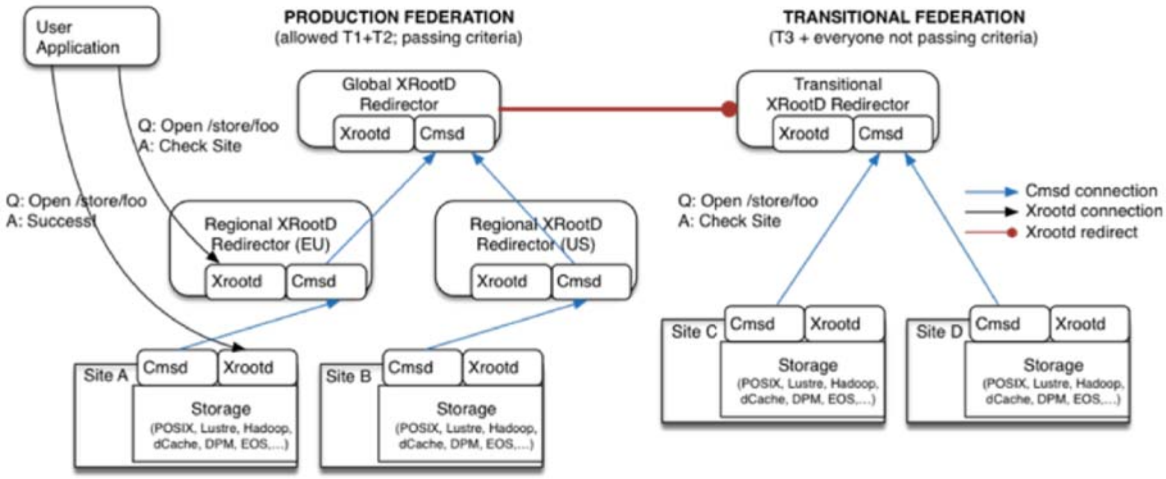


Figure 1. AAA, Any data, Anywhere, Anytime is built for the CMS experiment using the distributed CMS data cache. The resulting system, a hierarchy of redirectors, improves the scientific productivity of CMS physicists through better access to the data as well as effectiveness of the storage infrastructure deployed among the CMS sites on the worldwide grid.

The aim of these tests is to provide information for the user community and site administrators with respect to what to expect in terms of CPU efficiency and network usage for jobs accessing data remotely over the wide area network.

2. AAA and CMSSW infrastructure

2.1. Implementation of Any Data, Any Time (AAA)

Figure 1 shows a schematics of the XRootD infrastructure that spans all of the CMS Tier-1 and Tier-2 sites in Europe and the United States. The hierarchical subscription of XRootD managers (cmsd process) provides resiliency in data access towards clients. Effectively this means that, if the accessed data are not available at given site where the job lands, the client will be automatically redirected to the closest data available within the hierarchy of redirectors to the storage server where those data exist.

Each sites XRootD server is interfaced with the local storage system, allowing it to export the CMS namespace current storage systems (dCache and proxy dCache, HDFS, Lustre and DPM, EOS). Site servers also subscribe to a local redirector. In their turn, the local redirectors from each site then subscribe to a redundant regional redirector. This creates a large tree-structured federated storage system:

(i) United States: FNAL and Nebraska (DNS round-robin alias cmsxrootd.fnal.gov)
(ii) Europe: Bari, Pisa, Paris (DNS round-robin alias xrootd-cms.infn.it)
(iii) Transitional: CERN (DNS round-robin alias cms-xrd-transit.cern.ch)

Individual users or grid jobs can request a file from the regional redirector, which will then query the child nodes in the tree and redirect the user to a server that can serve the file. The entire AAA infrastructure overlays on top of existing storage systems, allowing users access to any on-disk data without explicit knowledge of the file location.

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 062042 doi:10.1088/1742-6596/898/6/062042

2.2. The CMSSW Application Framework

The overall collection of software of the CMS experiment, referred to as CMSSW, is built around a Framework, an Event Data Model (EDM), and Services needed by the simulation, calibration and alignment, as well as reconstruction modules that process event data used for physics analysis. The primary goal of the Framework and EDM is to facilitate the development and deployment of reconstruction and analysis software.

The CMSSW event processing model consists of one executable, called cmsRun, and many plug-in modules which are managed by the Framework. All the code needed for event processing (calibration, reconstruction algorithms, etc.) is contained in the modules. The same executable is used for both experimental data and Monte Carlo simulations. The CMSSW executable, cmsRun, is configured at run time by a job-specific configuration file. This file defines:

- which data to use;
- which modules to execute;
- which parameter settings to use for each module;
- the order or the executions of modules, called path;
- how the events are filtered within each path;
- how the paths are connected to the output files.

The cmsRun processing framework is very lightweight. At the beginning of the job, only the required modules are loaded dynamically. As described in the name, the CMS Event Data Model (EDM) is centered around the concept of an *Event*. An Event is a C++ object container for all RAW and reconstructed data related to a particular collision. During processing, data are passed from one module to the next via the Event, and are accessed only through the Event. All objects in the Event may be individually or collectively stored in ROOT files, and are thus directly browsable in ROOT. This allows tests to be run on individual modules in isolation. The auxiliary information needed to process an Event is called Event Setup and is accessed via the EventSetup object/class.

CMSSW reads the input 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. Caching is used for data access using the following three mechanisms.

(i) **Application caching:** This is the default method. It means that ROOT does the caching. If PoolSource.cacheSize is non-zero, a TTreeCache of that size is created per open file. Asynchronous read-ahead is turned off and the cache is filled with normal reads.

(ii) **Storage caching:** ROOT drives the caching using a prefetch list, but does not allocate a cache of its own. ROOT hands over the prefetch list to the storage layer, which is expected to do its own caching.

(iii) **Lazy-download caching:** Remote files are downloaded to a local shadow file on demand in 128MB segments. ROOT reads are directed to this local file. ROOT never reads directly from the remote file.

3. Implementation of the HTTP Data Access

3.1. External software used

In addition to CMSSW, the following software was also used for the development HTTP data access and for doing the comparisons with efficient data transfer protocols.

- Apache [12] is a widely used web server software. Developed and maintained by the Apache Software Foundation, Apache is an open source software available for free.

- Nginx [13] 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.
- FDT [14] 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 on all major operating systems and it is easy to use.
- The DaviX project [15] aims to make file management over HTTP-based protocols simple. The focus is on high-performance remote I/O and data management of large collections of files. Currently, there is support for the WebDav, Amazon S3, Microsoft Azure, and HTTP protocols. DaviX is multi-platform, open source and written in C++.

3.2. Implementation details

HTTP file access in CMSSW was already possible through curl, which downloads all files to be transferred to a scratch directory but it does not use RAM to accelerate the process. In CMSSW, Curl supports only the lazy-download mode. The new CMSSW plugin described here allows to:

- use x509 user certificates, VOMS proxy and RFC proxy with VOMS extensions;
- support read, vector read and vector read with offset;
- control the DaviX debug mode;
- access files from AWS S3, Microsoft Azure and WebDav.

The new plugin implemented in CMSSW substitutes the previous curl download method with the DavixAdaptor plugin [16]. During the implementation several issues had to be dealt with that are listed below.

- Until version 0.6.2 the DaviX library allocates multi-range buffers on the stack. If the chunk size is bigger than the stack size, DavixAdaptor plugins segfaults;
- Some DaviX debug messages were not correctly formated and show misleading information;
- The DaviX preadvec function does not return the actual number of bytes requested by the user but the number of data bytes that a call received from the server. In our case several of the ranges can overlap with the range of 10 bytes an DaviX will coalesce on overlapping ranges. This is done in order to limit the number of requests sent to the server with the aim of improving performance.
- In some cases and long lived TCP connections, if there is a dropped connection it causes a segfault in the preadvec function.
- Whenever DaviX is setting the log level, the underlying global variable being set is not atomic. This is not technically thread safe and we protected it to make sure it is only called once. The Davix team developers reported that issue was fixed also on their side.

All the above issues were reported back to the DaviX developers and they are already fixed or will be fixed in the next releases.

4. Performance tests and results

For initial performance tests and comparison of the different plugins we set up a testbed at Caltech:

- Data Node: Centos 7, RAM: 24 GB, Disk: 60TB RAID 0, Intel(R) Xeon(R) CPU X5670@2.93Ghz
- Execute Node: CERN Scientific Linux 6, RAM: 64GB, Disk: 4TB RAID 0, Intel (R) Xeon(r) CPU E5-2670@2.60Ghz

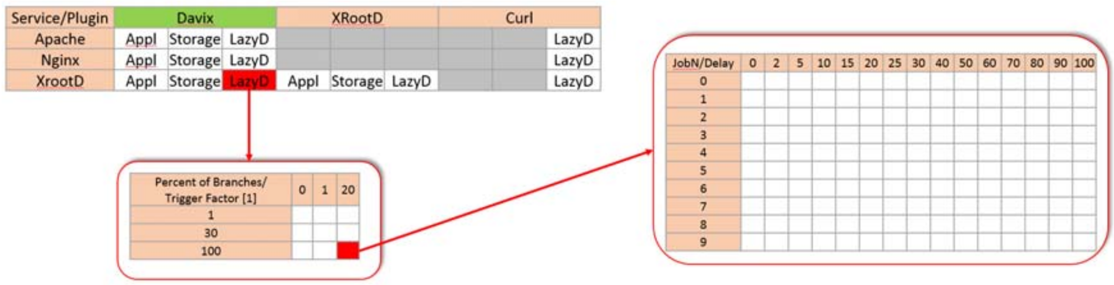


Figure 2. Test-workflow diagram. The table in the upper left shows all the different combinations of service and plugins as well as the configurable cache options for them. For each configurable option. The I/O adaptors are: 1) *Appl*: Application caching, 2) *Storage*: Storage caching, 3) *LazyD*: Lazy download. In total 15525 tests were performed with all the different variations of RTT, caching, plugin, percentage of branches and trigger factor.

- A default installation was performed, without modifications on the OS or other software. Between the nodes 1 network hop was configured. The NIC MTU was set to 9000 and txqueuelen was set to 50000.

The performed tests are shown schematically in Figure 2. A more detailed description is given below.

- For each test, only one job was run on the execute node, which read data from the data node using a specific protocol.
- Each job was retried up to 10 times and the average value of the results is used. This reduces statistical fluctuations as well as systematic ones due to specific conditions when running each test.
- Our tests use the IOExerciser and read a fixed percentage of branches from a file. At a fixed interval the tests read out all the data in an event. To emulate a bigger variety of jobs, a total 9 groups were selected of the percentage of branches/trigger factor. They emulate production, analysis and problematic I/O jobs.
- Tests are run with all possible I/O adaptors, like Application caching, Storage caching, Lazy-download.
- After each iteration of all tests, the data node network card was reconfigured to delay all packets (`tc qdisc add dev eth0 root netem delay 100ms`) and all tests are re-run again. This changes the effective RTT time for the data movement.
- Also, after each iteration of tests, the performance of the node and the throughput between nodes were tested to make sure there are no limitations on the hardware side.

The results are summarized in the tables and plots in Figures 3 and 4. In both figures the vertical axis shows the achieved throughput, while the horizontal axis is the RTT time. The tables in each figure contain the throughput number plotted with the best throughput for each case highlighted in red.

For Figure 3, the different curves show the performance for different transfer tools: xrdcp (with 4 and 8 transfer streams), FDT (with 2 and 15 transfer streams) as well as curl. FDT is seen to scale better with increasing RTT time.

For Figure 4, the different curves show the different caching mechanisms for CMSSW. The tests show no advantage in using storage caching and that lazy download is almost as efficient as application caching.

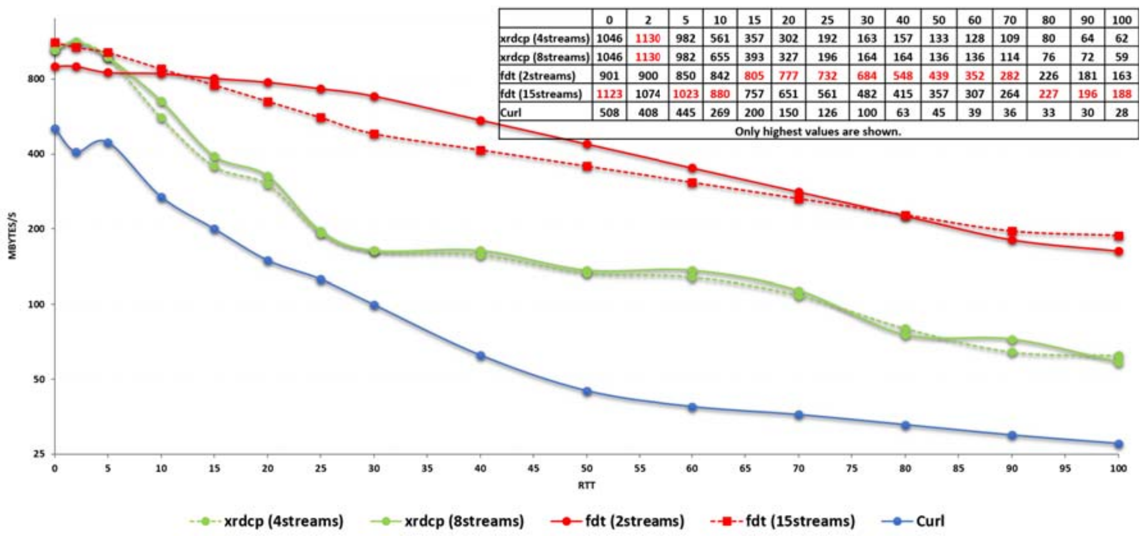


Figure 3. Performance network tests between the data and execute nodes with different round trip times. The vertical axis shows the throughput in MB/s, while the horizontal axis the RTT. The tests were performed with xrdcp, fdt, and curl using different numbers of streams. The tests show that FDT scales better than the other tools with increasing RTT.

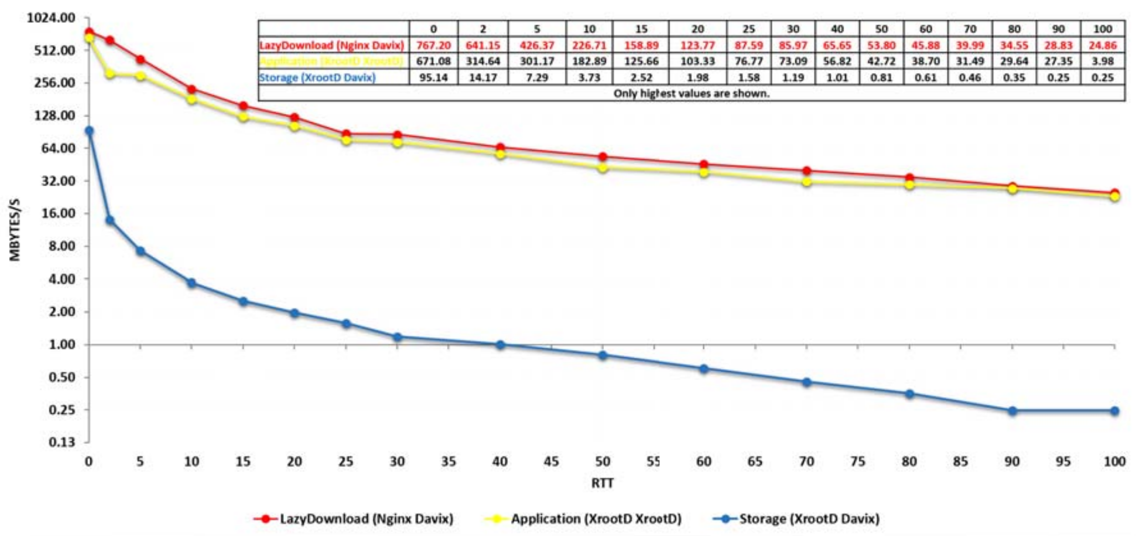
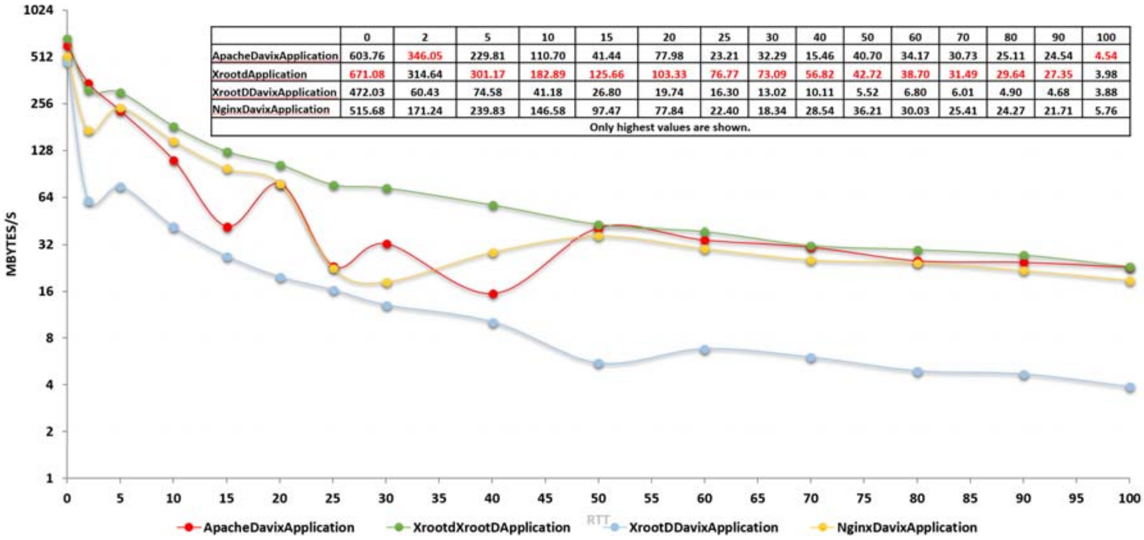


Figure 4. A comparison of the different CMSSW caching mechanisms. The vertical axis shows the throughput in MB/s, while the horizontal axies the RTT. The tests show no advantage in using Storage caching and that Lazy-download is almost as efficient as Application caching.The performance of HTTP data access over XrootD is still poor.



	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.															

Figure 5. Comparison of Application caching between XRootD and other HTTP protocol providers.

Figure 5 shows application caching performance for a different round trip time of the XrootD protocol and Davix plugin. Data are read through Apache, NgINX, and HTTP over XrootD. These tests show that application caching performance is similar for XrootD and the Davix plugin reading from Apache or Nginx. The tests also show a suspiciously (constant) low performance when reading through HTTP over XrootD. Further investigation is ongoing to find out what causes this bottleneck.

5. Summary

A first version of the HTTP DaviX plugin for CMSSSW is implemented. A variety of tests was done using the different caching mechanisms as well as different protocols. Overall, choosing the right caching algorithm will lead to better job efficiency and increased network usage. Each job requires a different amount of data that is used for analysis. Therefore, CMS tools that run the workflows should decide on the caching mechanism based on the jobs need for data and RTT. For example, if RTT is small, aplication and lazy download will perform the same. The initial measurements described above can be used as orientation for such decisions.

Development is ongoing to implement S3 and WebDav support. Furthermore, we are collecting and analyzing metrics of of CMS production and analysis jobs. This information will be used to further improve job efficiency.

Acknowledgments

The authors would like to acknowledge the support of the software and computing program of the CMS experiment at the LHC.

References

[1] Bird I, Computing for the Large Hadron Collider. Annu. Rev. Nucl. Part. S. 2011; 61: 99.
[2] CMS Collaboration 2008 JINST 3 S08004
[3] Aad G et al. 2008 ATLAS Experiment at the CERN Large Hadron Collider. JINST 3 S08003.
[4] Alves, A et al. 2008 The LHCb Detector at the LHC, JINST 3 S08005
[5] Aamodt K et al. 2008 The ALICE experiment at the CERN LHC, JINST 3 S08002

[6] Stewart G A and McCance G 2006 Grid Data Management: Reliable File Transfer Services Performance (Mumbai, India: CHEP06)
[7] XRootD project page: http://www.xrootd.org/
[8] CMS Xrootd Architecture, https://twiki.cern.ch/twiki/bin/view/Main/CmsXrootdArchitecture
[9] CMS Software framework, http://cms-sw.github.io/
[10] Amazon Simple Storage Service, http://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html
[11] WebDav, http://www.webdav.org/
[12] Apache, https://www.apache.org/
[13] NGINX, https://www.nginx.com/resources/wiki/
[14] FDT, http://monalisa.cern.ch/FDT/
[15] Davix project webpage: https://dmc.web.cern.ch/projects/davix/home
[16] Pull request to CMSSW, https://github.com/cms-sw/cmssw/pull/15130