Progress on Machine and Deep Learning applications in CMS Computing

D. Bonacorsi[*]
University of Bologna, INFN-Bologna
E-mail: daniele.bonacorsi@unibo.it

V. Kuznetsov

Cornell University

L. Giommi

University of Bologna, Italy

T. Diotalevi

University of Bologna, Italy

J.R. Vlimant

Caltech, United States

D. Abercrombie

MIT, United States

C. Contreras

DESY, Germany

A. Repečka

Vilnius University, Lithuania

Z. Matonis

Vilnius University, Lithuania

K. Kančys

Vilnius University, Lithuania

Machine and Deep Learning techniques are being used in various areas of CMS operations at the LHC collider, like data taking, monitoring, processing and physics analysis. A review a few selected use cases - with focus on CMS software and computing - shows the progress in the field, with highlight on most recent developments, as well as an outlook to future applications in LHC Run III and towards the High-Luminosity LHC phase.

International Symposium on Grids and Clouds 2018
16-23 March 2018
Academia Sinica, Taipei, Taiwan

[*]Speaker.

1. Introduction

The CMS Computing system performed smoothly in LHC Run-1/2. In its evolution towards Run-3 and beyond, it would largely benefit from some more detailed modelling of workflows, subsystems performances, site and infrastructure behaviours. The exploitation of such modelling would allow to make predictions, hence adding more automation into the existing systems, and ultimately adaptive behaviours in real operations. These efforts started since few years, and a refinement of the techniques used, as well as the adoption of such approaches to previously unexplored sectors of CMS computing, is evident. This contribution aims at briefly summarising the state of the activities in some (selected) areas, and to assess the next steps.

With respect to just few years ago, the Computing (meta)data - by which we do not refer to collision data or calibration data, but to the vast amount of monitoring and logging information available from CMS computing operations in both the workload and data management sectors - have started to be proficiently explored. This happened e.g. with data about transfer operations, replication performances, dataset accesses priorities, job submissions patterns, job resubmission actions, site performances and operations tuning, infrastructure and services behaviours, and more. In all cases, the goal is to extract actionable insight from this data through relatively small-scale projects, each focusing on a specific problem and developing new “tools” and prototypes to extract value from the data and transform this value into actions to be eventually streamlined in CMS computing operations. In other words, a “data science” approach is growing to study and exploit such CMS computing (meta)data. Collaboration with computer scientists and data scientists out of HEP are being enforced, so to help LHC computing experts to break jargon barriers by learning to formulate problems to non-HEP experts, and ultimately to build synergies across different scientific communities.

2. Dataset popularity

The so-called “popularity” of CMS datasets [1] has historically been one of the areas in which machine learning techniques have been pioneered in CMS computing. Different metrics can be chosen to define how “popular” a CMS dataset is for distributed analysis users, e.g. number of accesses to a dataset in a given time unit, number of users accessing a dataset in that time unit, number of CPU-hours spent on analysing a given dataset, etc. Regardless of the best definition used, it can evidently be treated as a classification problem in supervised machine learning. The goal is to use the vast amount of access information from historical records to predict the popularity of new datasets appearing in the future. This might result in very concrete operational actions, yielding an evident computing resources optimisation (primarily on the storage side), e.g. optimise the number of dataset replicas worldwide, by reducing the number of replicas of datasets predicted to be unpopular, or viceversa increasing the distributed copies of datasets predicted to become popular in the future.

The approach followed these steps. Firstly, detailed studies of the impact of one popularity definition versus others was performed [2], in order to assess which was the definition that yielded the highest (and stable) true positive rate: the latter was one the most important metric, as the identification of a popular dataset in the future would trigger a costly action in operations, i.e. its

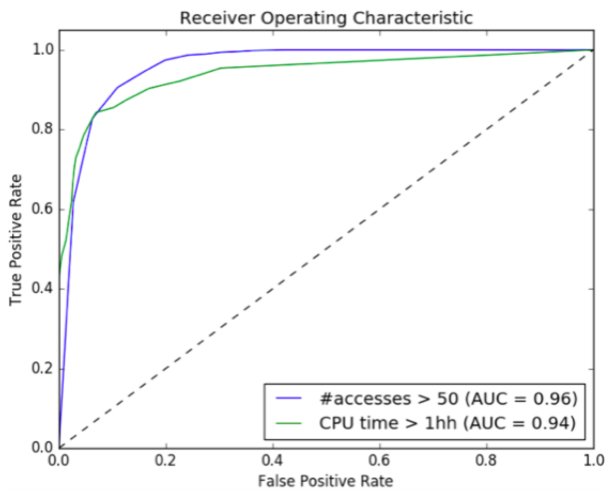

Figure 1: Area under the ROC curve for the machine learning model used in the CMS dataset popularity prediction, corresponding to two different metric definitions.

replication on storage systems worldwide - while the false positive rate just need to be kept reasonably low (e.g. 1% of false positive rate - based on hundreds of newly created datasets and given their average size is roughly 2 GB per datasets - would only cost CMS extra 10 TB of data transfers per week). A good definition of what is "popular" was settled via rule-based combined criteria, exploiting the number of accesses to a dataset and the number of CPU-hours spent analysing it. Secondly, CMS (and non-CMS) data services were queried to acquire the data (services like PopDB, DBS, SiteDB, PhEDEx, Site Support tools were used, as well as LHC Dashboard) [3, 4]. Thirdly, at any given time, data about previous months' accesses were used to deliver popularity predictions for the following week. A "rolling" window approach was followed, i.e. N months of data in the past were used to train classifiers (a good working point was found at N=6), prediction for the following week were produced, and then this week was added to the training set for subsequent re-training, etc [5]. The work started from millions of data frames extracted from (primarily) CMS data services in the 2013 - mid 2015 period, and analysed. The effect of the popularity definition on different CMS data types was also performed, showing that the predictive potential on known-to-be-needed datasets was solid [4].

The study of various classifiers showed that e.g. gradient boosting techniques (such as XGBoost [6]) offered a comparably higher performance, despite at a relatively higher computational cost. As the choice of the underlying infrastructure and setup for the machine learning task was evidently crucial, it was decided to migrate all the modelling part to Apache Spark [7], exploiting the CERN-IT HDFS cluster. The CMSSpark framework [8], developed in CMS to unlock the Spark potential to CMS users, was used. All the original code base was rewritten to run in Spark, and 3 classifiers that are available in Spark+MLlib [9] - namely RandomForest [11], DecisionTree [**?**], GBT [12] - were used. Further optimisations and continued efforts in this task allowed CMS to reach an accuracy (AUC) of 96% in predicting popular datasets [13], see Figure 1.

This work is ready to be seeded into next generation of CMS dynamic data management system, which is under discussion in these months in the CMS Computing community. Any new

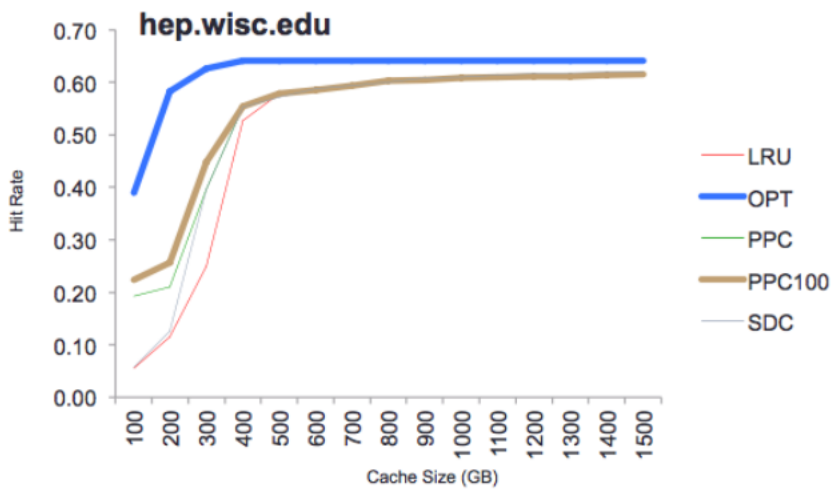

Figure 2: Example of how popularity-based predictions might be input to strategic cache management at sites. Details in the text.

system that will be able to exploit this predictive engine will streamline data replication tactics and eventually lead to optimised site (storage) utilisation by construction. On the other hand, for this to be concretely useful today already, it is not mandatory that it is integrated into the CMS global data management system: every CMS Grid site can already use this information as of today, and at zero cost. An example is shown in Figure 2 and discussed in the folowing. The popularity predictions can be applied - if not to the overall CMS - just to the dataset placement on a specific site: at a Grid site a site administrator may well learn (according to various algorithms, e.g. LRU, OPT, PPC, SDC, etc) which is the expected hit rate on the storage cache in the next months, thus allowing to make strategic choices (e.g. do not evict cache elements if predicted to be popular for next N weeks), and ultimately decide which is the optimal cache size for their site depending on the predicted hit rate [13].

3. Transfer latencies

The transfer latency use case refers to predicting how long it would take to transfer different datasets of different sizes across a complex worldwide topology of WLCG [14, 15] computing Tiers. The transfer latencies may impact the CMS analysis and production workflows in various ways, also very heavily: any technique that would help to classify them into latency classes, for example, up to precisely predict a transfer completion time, could make a difference, as this insight might be enforced in the routing logic of any experiment-level data transfer tool. In this sense, machine learning technique are particularly promising, and the problem might be formulated as classification or regression, depending on the strategy.

The base ingredients for this project are the transfer logs from PhEDEx [16, 17] - the current CMS reliable and scalable dataset replication system - and FTS [18] - the File Transfer Service used by LHC experiments. Preliminary and preparatory work started since long time ago, when CMS PhEDEx had been equipped to save the relevant data indefinitely on persistent storage [19]. Subsequently, exploratory analyses on various latency types and signatures have been performed, skew variables has been used to describe them properly, etc [20]. The reasons for a transfer issues might vary a lot, from transient storage issues, to corrupted files, to network problems, etc. This work lead to the identification of transfer clusters/categories, e.g. the transfer that got stuck early in the process for some reason (i.e. "early stuck", transferring only a small fraction of the total size and then stopping or slowing down) or late in the process (i.e. "late stuck", smoothly transferring the majority of the total size but then stopping or slowing down and failing to finish). While early stuck transfers can easily be cancelled by operation teams in favour of trying different source files, late stuck transfers have already used network bandwidth to move most of the dataset content, but this is useless unless the transfer is fully complete so that CMS analysts can access the transferred data. The identification of these cases well in advance may help operation team to cure problematic transfers, and take proper actions in due time: ultimately, for most clear signatures automatic procedures can be set up.

This project articulated in various steps. Firstly, the transformation of PhEDEx/FTS data into a machine learning suitable form is necessary. FTS databases host TBs of raw data, that have been collected, cleaned and injected into a CERN HDFS cluster. JSON objects have been converted from ASCII files to a flat table format (CSV), including dealing with all peculiarity in the original format (deal with special characters and custom EOF, nested records to be flattened, placeholder manually set for all missing attributes, etc), then hashing algorithms have been used to convert text into numerical values [21]. More data preparation steps were taken, e.g. *i)* attributes about the end of a transfer (e.g. failure occurrences, transfer Δt) cannot be used for predictions and should be removed; *ii)* attributes that are static through all datasets are uninformative, waste space and even may mislead algorithms, so they should be dropped; *iii)* select one among obviously correlated attributes (e.g. number of files vs file size, given the roughly stable average size of the vast majority of CMS files) [22]. All these steps yielded a reduction to about 50% of the original CMS dataset [21, 22].

Also in this use case, a Spark [7] platform can solve many of the issues outlined above, and provide cost-effective solutions to prepare data for the application of machine learning. The CMSSpark framework [8] developed in CMS and used for data popularity has also been used in this case, to parse logs/data from about a dozen of CMS (and not) data-services, i.e. AAA, EOS, FTS, CRAB, CMSSW, HTCondor, DBS, PhEDEx, WMAgent, etc. In this use case, tree-based classifiers performed among the best ones. Focus has been on RandomForest and GradientBoost regressors from the scikit-learn [23] library, XGBRegressor from the XGBoost [6] library. The obtained results show that with this approach a 80% accuracy (at least) can be achieved [20], with more work in progress.

As a side note, it is remarkable that the deeper knowledge on transfer system behaviours that this project allowed to build was the basis for new projects that may help to properly tackle transfer congestions and thus reduce ETAs. An example of how transfer system router decisions could be developed in future data management tools with the exploitation of this insight can be see in "*transfer2go*" [24], a CMS R&D project in collaboration with the Google Summer of Code [25] student programme in Summer 2017.

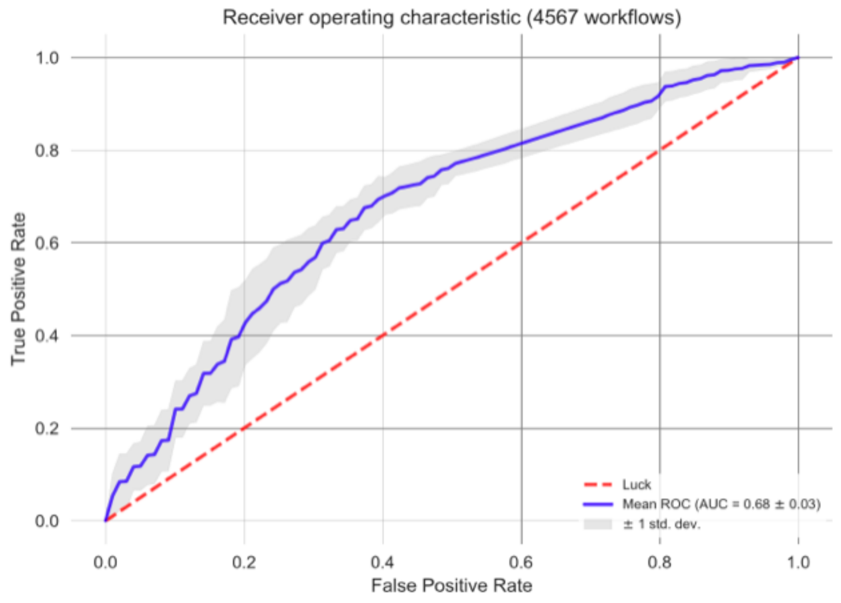

Figure 3: Area under the ROC curve for the first attempt of a deep learning model (with no optimisations yet) for the prediction of the optimal job recovering actions in the CMS workflow management sector.

4. Workflow handling

In the CMS Workflow Management sector, the work done by the operation team to manage centrally-orchestrated Grid/Cloud jobs in CMS is very time- and manpower-consuming. The goal of machine learning in this area is to deliver error handling predictions towards transforming the majority of manual operator intervention into automated actions.

The central CMS MC production system manages thousands of "workflows", each composed of thousands of jobs. Common issues in workflow management are errors in Grid jobs e.g. about missing/corrupt input files, high memory usage, etc. All these are currently handled manually by the operations team, i.e. an expert must look at the error codes on the monitoring systems and decide which actions have to be taken (e.g. job kill and resubmit, workflow invalidation, cloning, recreation, etc). While these operations are very costly on a large scale, it is also true that some error classes are easy to be identified, and obvious responses exist. An algorithm that encompasses all possible patterns that can be anticipated would be difficult to write and - most importantly - impossible to maintain. This is a use-case in which unsupervised machine learning techniques stand as a natural solution to explore. This method would allow to limit, group, enumerate possible operator actions, so that for well-defined groups of errors the operators could kill/clone/resubmit/recover the jobs, as appropriate.

The strategy foresaw an implementation in which unsupervised machine learning comes before operator intervention. Data is pulled into the system from CMS data services: currently errors thrown (from WMStats), site statuses (from SiteDB and CMS Site Support team), and in the next future also more workflow parameters (from Request Manager), job splitting information, data about usage of XRootD [26] or not, data about requested memory by jobs, etc. The data is basically organised into a single matrix, with the number of times each error codes occurs per site, times the site status at the time of the error occurrence, e.g. enabled, disabled, drain (in order to limit the complexity to a factor of 2 only, the latter site status was collapsed into "good" and "bad" site statuses). Then, the K-means clustering algorithm was used to group together similar error-type workflows together, in such a way that multiple workflows can then be acted on at the same time. In order to make the cluster characteristics stable, the stored workflow error patterns and site statuses over the past few months was included. More sophisticated techniques to compress errors and sites phase space in order to better group similar errors, and hence to fight against sparse matrices, are part of the future work plan.

An alternative supervised machine learning approach with deep neural network was also adopted in this area, with the specific goal to predict the CMS jobs recover (so-called "ACDC") actions versus resubmit (clone) actions. The input data is, also in this case, the same "error codes - site status" matrix information as described in the previous paragraph. A first implementation of such deep neural network (not optimised yet) has 5 layers, 75 neurons (hidden units) per layer, uses ReLU as the activation function, applies a dropout at 0.002, a learning rate of 0.001, uses the binary cross-entropy as loss function, and Adam for gradient descent optimisation. First results showed an accuracy (AUC) of $(68 \pm 3)\%$, as shown in Figure 3, without any ad-hoc optimisation efforts applied yet.

For this use-case, Keras [27] was used for deep neural network modelling; the scikit-learn [23] library was used to handle the model building pipeline, to apply feature scaling, resampling, andÊ-modelling; Pandas [28] was used to store input data as data frames. Matplotlib [29] was used for plotting, and Scipy [30] was used to handle computation such as KS-test and p-value.

5. Event classification

The goal of event classification is to classify which HEP event occurred in a collision, based on the data at disposal, coming from all subdetectors. Needless to say, this is an extremely crucial and delicate use-case: novel approaches are trying to infer the event type from high-level objects identification via image classification techniques. For example, one possible approach to this task is the application of convolutional neural networks (CNN) to classify events instead of using e.g. traditional tracking algorithms. This is another example of application of unsupervised machine learning algorithms. The ultimate goal in this case is to explore novel ways to perform this task within computing resources budget constraints, a goal that will become more and more critical towards the High Luminosity LHC phase.

A possible method can be briefly outlined as follows: CMS pixel/silicon hits are extracted in global coordinate frame, then transformed into PNG images (within a single CMSSW EDAnalyzer), then fed to a CNN with a proper architecture (see Figure 4 for a pictorial view). By training with a relatively small sample of 27K images, and in a set-up that allows to use world-class models and algorithms (e.g. via fast.ai [31], a framework based on PyTorch [32]), very good accuracies can be obtained, e.g. on average higher than 90% in classifying J/Psi events versus QCD, or Higgs events versus J/Psi. A full description of this use case and its recent advancement goes beyond the scope of this paper, but some technical aspects of this work are remarkable and worth a mention, as they stand as key technical point for the advancement of the application of machine learning methods to HEP use-cases: two specific remarks are outlined in the following two paragraphs.

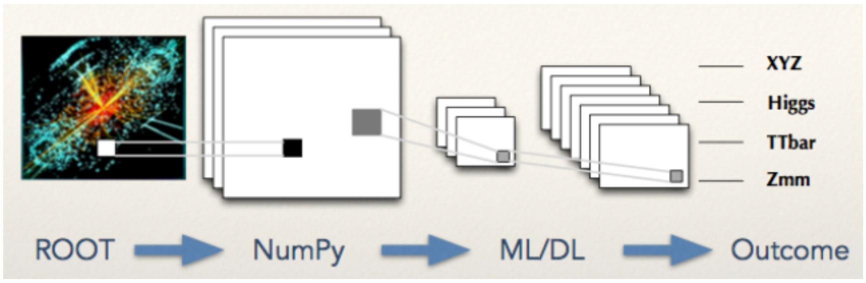

Figure 4: Pictorial view of the event classification use case. Details in the text.

As a first remark, DIANA-HEP [33] - an initiative funded in US by the National Science Foundation and aiming at developing state-of-the-art software tools for HEP experiments - launched the "*uproot*" project [34]. *Uproot* provides native access to ROOT IO without any particular knowledge of the ROOT file structure. It is implemented as a pure Python ROOT reader that directly copies columnar ROOT data into (for example) NumPy arrays. This is a key step in machine learning application, as it offers a reliable and scalable method to transform massive amount of classical HEP data in ROOT format into file format (e.g. CVS) that are standard for machine learning applications worldwide, thus unlocking the power of world-class frameworks and tools also on HEP data. For example, a loop over events is not implemented as a (slow) Python loop, but as (fast) compiled code with NumPy arrays behind, and iteration in batches foreseen for big datasets. Actually, for extremely big datasets, things get even better with Spark, as it is now possible to read in ROOT files on Spark platform natively.

As a second remark, it is being acknowledged that performing machine/deep learning in production at scale is a challenging task. Pushing a HEP machine learning based project beyond a basic exploratory phase is not trivial: we saw in the previous paragraph the issues related to reading ROOT files efficiently, but other challenges can be identified, such as the difficulty to profit of adequately scaled infrastructures for training, as well as the proper data science skills needed in most R&D activities (which are sometimes hard to find in the HEP environment, as they are sometimes complementary to the skills needed for HEP analyses). The idea of taking all data transformation steps, and streaming to emerging frameworks (e.g. Google Tensorflow [35]) as well as allowing access to solid hardware resources for training (e.g. GPUs, HDFS cluster, ..) can be entirely designed "as a service" for many use-cases. This approach envisions "machine learning as a service" as a viable approach to find a synergy between the potential of these techniques for HEP analyses and the vast amount of computing resources potentially offered by Cloud computing. In this area, CMS pioneered an effort towards "Tensorflow as a service" (TFaaS) [36], an end-to-end data-service able to read ROOT files and convert them into machine/deep learning input, serve machine/deep learning models (prepared with Tensorflow or Keras+Tensorflow, for example) via REST API, exchange data via high efficient transport layer (e.g. proto-buffers), read data remotely and integrate service calls into the CMSSW software stack, together with the inherent ability to deploy such service on the Cloud (e.g. rent GPUs to train models). A fully working prototype is ready [38], and a demo was shown by the speaker at the conference. A full description of its technical details is in preparation in an ad-hoc paper. The open source code can be found at [36], with a docker image also available at [37]. It has been demonstrated as a proof of concept for 2 very different use-cases: *i)* event classification (as described above); *ii)* signal versus background discrimination in all-hadronic top decays in CMS [38].

6. Conclusions

Machine learning and deep learning are rapidly evolving approaches to characterising and describing data with the potential to radically change how data is reduced and analysed, also at LHC. The CMS experiment is exploring the adoption of such techniques in various areas: in particular, this paper focussed on summarising the most recent advancements in (selected) projects in the CMS software and computing domain. From data management (dataset popularity and transfer latencies) to workload management (automatic job management tactics), CMS Computing shows an interestingly wide range of successful applications, some of which are production ready. CMS Computing is increasing and extending its efforts in adopting machine learning techniques to more use cases, aiming to extract more and more actionable insight from the richness of the computing logs and (meta)data collected in Run-1/2, and aims at building next-generation adaptive systems in CMS computing.

References

[1] D. Giordano et al, "*Implementing data placement strategies for the CMS experiment based on a popularity model*", J.Phys.Conf.Ser. 396 (2012) 032047

[2] D. Bonacorsi et al, "*Exploiting CMS data popularity to model the evolution of data management for Run-2 and beyond*, 2015 J. Phys.: Conf. Ser. 664 032003, http://iopscience.iop.org/article/10.1088/1742-6596/664/3/032003/pdf

[3] K. Kancys, CERN Summer Student 2016 report (on github: http://cern.ch/go/8lGQ)

[4] V. Kuznetsov et al, "*Predicting dataset popularity for the CMS experiment*, 2016 J. Phys.: Conf. Ser. 762 012048, http://iopscience.iop.org/article/10.1088/1742-6596/762/1/012048/pdf

[5] DCAF framework, https://github.com/dmwm/DMWMAnalytics/tree/master/Popularity/DCAFPilot

[6] XGBoost, https://xgboost.readthedocs.io/en/latest/

[7] http://spark.apache.org/

[8] CMSSpark, https://github.com/vkuznet/CMSSpark

[9] https://spark.apache.org/mllib/

[10] https://spark.apache.org/docs/latest/mllib-decision-tree.html

[11] https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.tree.RandomForest

[12] https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.tree.GradientBoostedTrees

[13] "*Dataset Popularity Prediction for Caching of CMS Big Data*", J Grid Computing (2018) https://doi.org/10.1007/s10723-018-9436-4

[14] J. D. Shiers, "*The Worldwide LHC Computing Grid (worldwide LCG)*", Computer Physics Communications 177 (2007) 219–223, CERN, Switzerland

[15] WLCG web portal, http://lcg.web.cern.ch/lcg/

[16] T. Barrass et al, “*Software agents in data and workflow management*”, Proc. CHEP04, Interlaken, 2004. See also http://www.Þpa.org

[17] R. Egeland et al., “*Data transfer infrastructure for CMS data taking*”, XIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT’08), Erice, Italy, Nov 3-7, 2008 - Proceedings of Science, PoS (ACAT08) **033** (2008)

[18] File Transfer Service, https://egee-jra1-dm.web.cern.ch/egee-jra1-dm/fts/

[19] T. Chwalek et al, “*No file left behind - monitoring transfer latencies in PhEDEx*”, J. Phys. Conf. Ser. 396 (2012) 032089, http://iopscience.iop.org/article/10.1088/1742-6596/396/3/032089/pdf

[20] D. Bonacorsi et al, “*Monitoring data transfer latency in CMS computing operations*”, J. Phys. Conf. Ser. 664 (2015) 032033

[21] Z. Matonis, CERN Summer Student 2016 report (on cds.cern.ch: http://cern.ch/go/7RXN)

[22] T. Diotalevi, CERN Summer Student 2016 report (on cds.cern.ch: http://cern.ch/go/9hmp)

[23] Scikit-learn, http://scikit-learn.org/

[24] transfer2go, https://github.com/vkuznet/transfer2go

[25] Google Summer of Code, https://summerofcode.withgoogle.com/

[26] XRootD, http://xrootd.org

[27] Keras, https://keras.io/

[28] Pandas, https://pandas.pydata.org/

[29] Matplotlib, https://matplotlib.org/

[30] Scipy, https://www.scipy.org/

[31] fast.ai, http://www.fast.ai

[32] PyTorch, https://pytorch.org/

[33] DIANA-HEP, http://diana-hep.org/

[34] uproot, https://github.com/scikit-hep/uproot

[35] Google Tensorflow, https://www.tensorflow.org/

[36] TensorFlow as a Service, https://github.com/vkuznet/tfaas

[37] TFaaS on docker, https://hub.docker.com/r/veknet/tfaas/

[38] “*Prototype of Machine Learning "as a service" for CMS Physics in Signal vs Background discrimination*”, L. Giommi, Master thesis, University of Bologna