ISGC'18 - Taipei (ASGC)

Progress on Machine and Deep Learning applications in CMS Computing

V. Kuznetsov (Cornell Univ., US)
D. Bonacorsi , L. Giommi, T. Diotalevi (Univ. Bologna / INFN)
J.R. Vlimant (Caltech, US)
D. Abercrombie (MIT, US)
C. Contreras (DESY, Germany)
A. Repečka, Z. Matonis, K. Kančys (Univ. Vilnius, Lithuania)

(on behalf of the CMS experiment)

Outline

Introduction

Experience on:

- **dataset popularity**
- **transfer latencies**
- **workflow handling**
- **event classification**, and prototype of **ML as a service**

Summary

Look for these tags to find more in the references: *Refs: [X]*

Introduction

Refs: *[1]*

Actionable insight and adaptive modelling

The CMS Computing system worked nicely in LHC Run-1/2. In its evolution towards the future, it largely benefits from:

- some detailed **modelling** of workflows / systems performances / site behaviours
- the exploitation of such modelling to make **predictions**
- more automation, and **adaptive** behaviours in ops

With respect to just few years ago:

- the Computing (meta)data starts to be accessed and appropriately explored
    - e.g. data transfer operations, replication performances, dataset accesses priorities, job submissions patterns, job resubmission actions, site performances and operations tuning, infrastructure and services behaviours, …
- actionable insight is extracted
    - new "tools" and prototypes, in form of projects to exploit and streamline actions based on this insight
- a "data science" approach to study and exploit CMS computing (meta)data
    - at the same, an *opportunity* and a *challenge*
        - bridge communities, break jargon barriers, learn to formulate problems to non-HEP experts, ..

Dataset popularity[*]

[classification]

[*] Different metrics may define how "popular" a CMS dataset is for distributed analysis users, e.g. # accesses, # users, CPU-hrs, ..

Goal: computing resource optimisation (primarily storage)

Refs: [2-4]

Approach for popularity

Supervised ML (classification)

- define what's "popular" via rule-based criteria (e.g. #access > N)
- use CMS (and non-CMS) data services to acquire the data
  - CMS {PopDB, DBS, SiteDB, PhEDEx, Site Support tools}, but also e.g. Dashboard
- use previous months' accesses to deliver popularity predictions for next week

"Rolling" window approach

- N months of data in the past used to train classifiers
  - good working point found at N=6
- produce predictions, then add this week to the training set and re-train, etc

Samples and popularity metric definition

- Started from millions of data frames extracted from (primarily CMS) data services in the 2013 - mid 2015 period, and analysed

- Settled on settle on #access>10 as an adequate "popular dataset" metric

- Studied the effect of such cut on different CMS data types

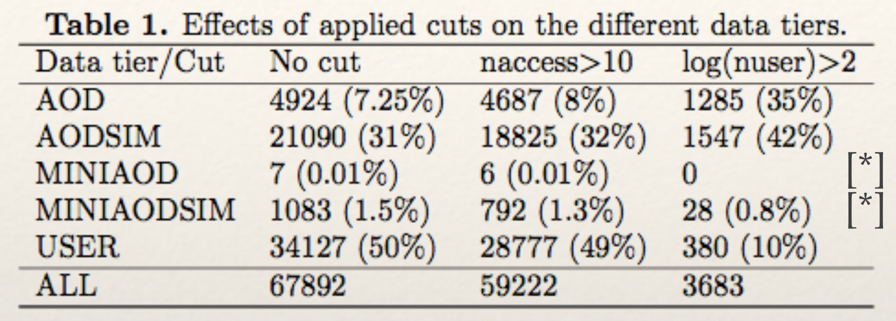
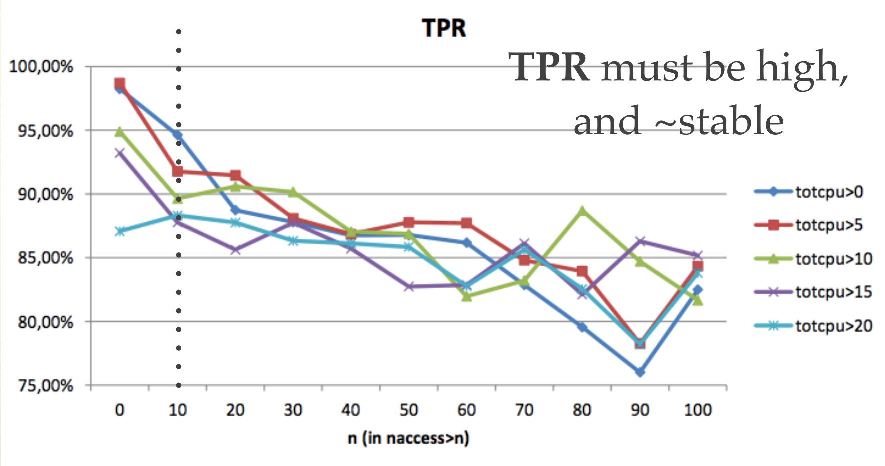


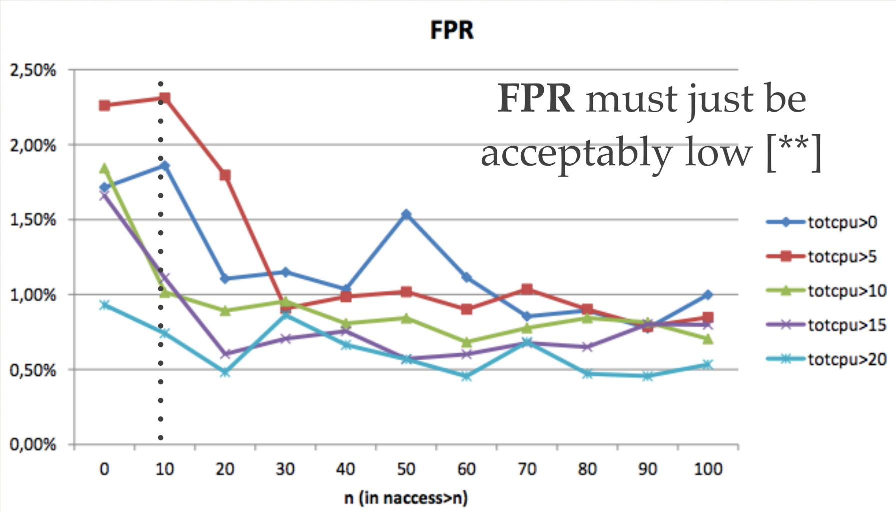
Table 1. Effects of applied cuts on the different data tiers.

Data tier/Cut	No cut	naccess>10	log(nuser)>2
AOD	4924 (7.25%)	4687 (8%)	1285 (35%)
AODSIM	21090 (31%)	18825 (32%)	1547 (42%)
MINIAOD	7 (0.01%)	6 (0.01%)	0 [*]
MINIAODSIM	1083 (1.5%)	792 (1.3%)	28 (0.8%) [*]
USER	34127 (50%)	28777 (49%)	380 (10%)
ALL	67892	59222	3683

[*] MINIAOD* introduced only in 2014



TPR must be high, and ~stable



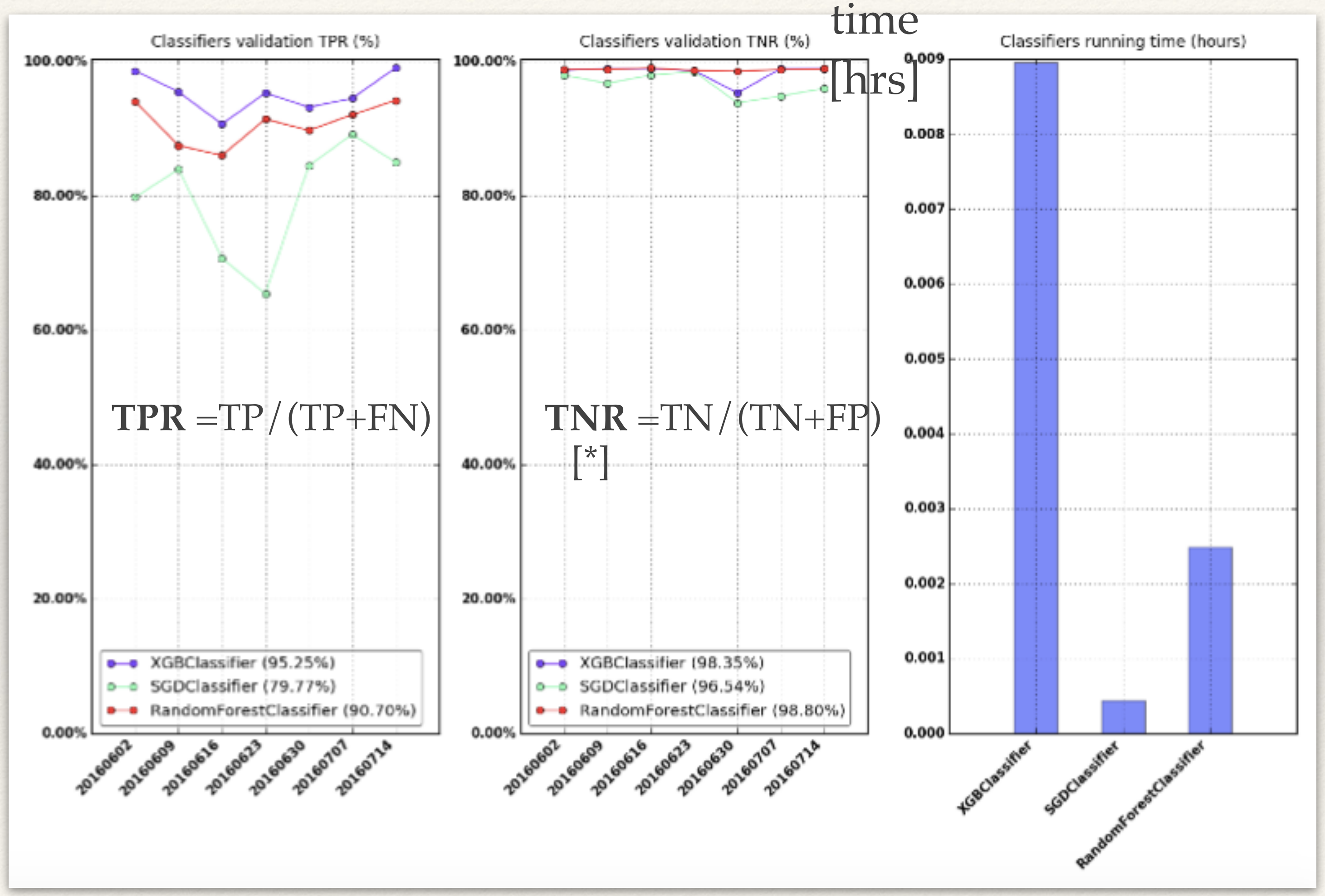
FPR must just be acceptably low [**]

[**] e.g. 1% of FPR - based on $\mathcal{O}$(hundreds) newly created DSs and size $\mathcal{O}$(2 TB)/DS - would cost CMS extra 10TB of data transfers / week

Phase 1: first predictions of CMS dataset popularity

Various classifiers were used, results differ mostly in performances only

- performance matters, but changes a lot with infrastructure and setup - see next for Spark



%

time [hrs]

TPR =TP/(TP+FN)

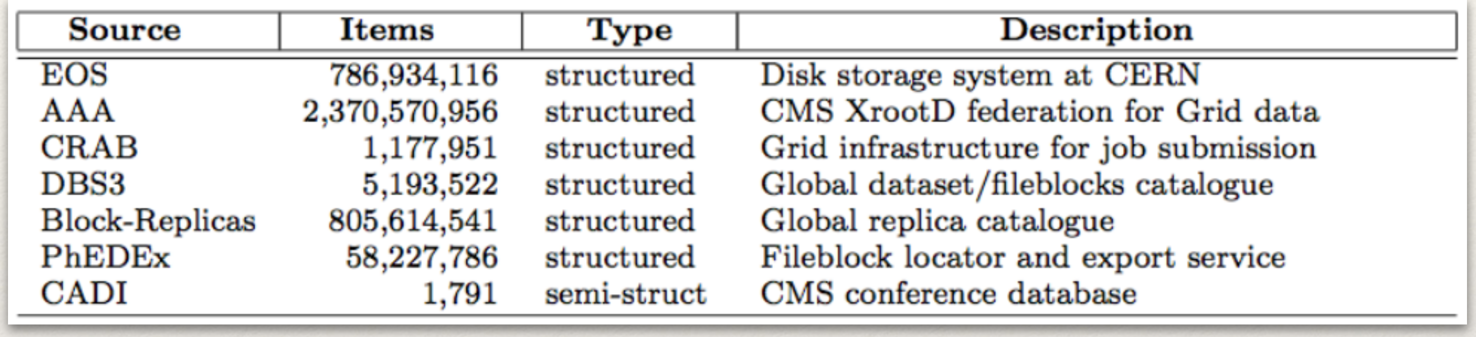
TNR =TN/(TN+FP) [*]

[*] e.g. 1% of FNR - based on $\mathcal{O}$(1.5M jobs/day in CMS - would correspond to O(15k) jobs to compete for resources where popular dataset may reside

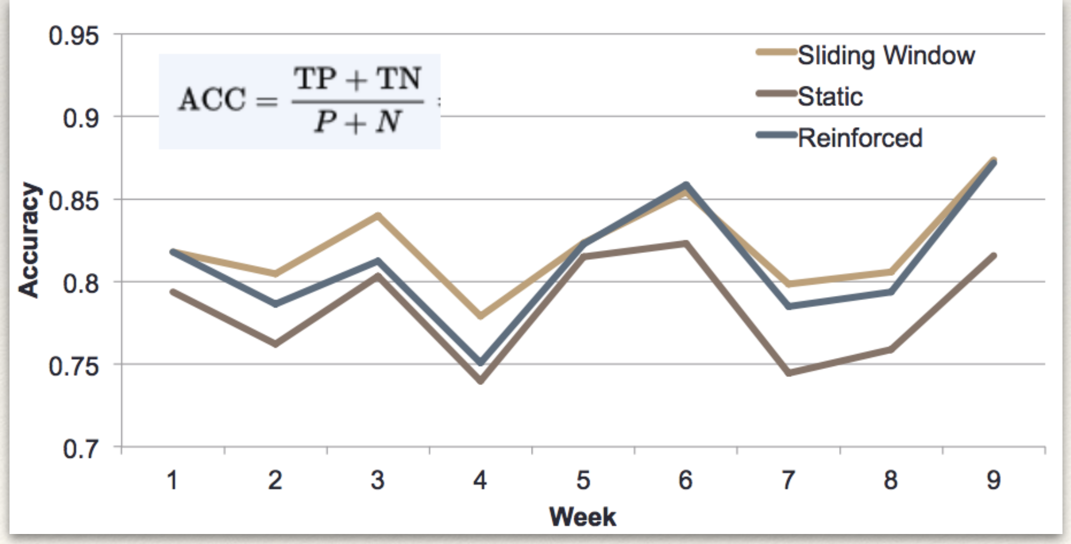
Phase 2: populating Spark (CERN-IT cluster)

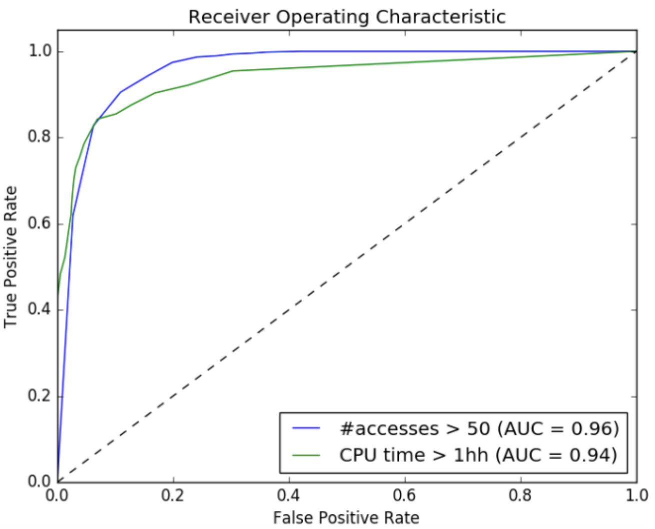
Decided to move the modelling part to **Apache Spark**

- exploitation of the CERN-IT HDFS cluster. **CMSSpark** framework developed and used. Code written to run in Spark using 3 classifiers available in *Spark+MLlib* (RandomForest, DecisionTree, GBT)



Source	Items	Type	Description
EOS	786,934,116	structured	Disk storage system at CERN
AAA	2,370,570,956	structured	CMS XrootD federation for Grid data
CRAB	1,177,951	structured	Grid infrastructure for job submission
DBS3	5,193,522	structured	Global dataset/fileblocks catalogue
Block-Replicas	805,614,541	structured	Global replica catalogue
PhEDEx	58,227,786	structured	Fileblock locator and export service
CADI	1,791	semi-struct	CMS conference database





Continued efforts reached an accuracy (AUC) of **96%** in predicting popular datasets

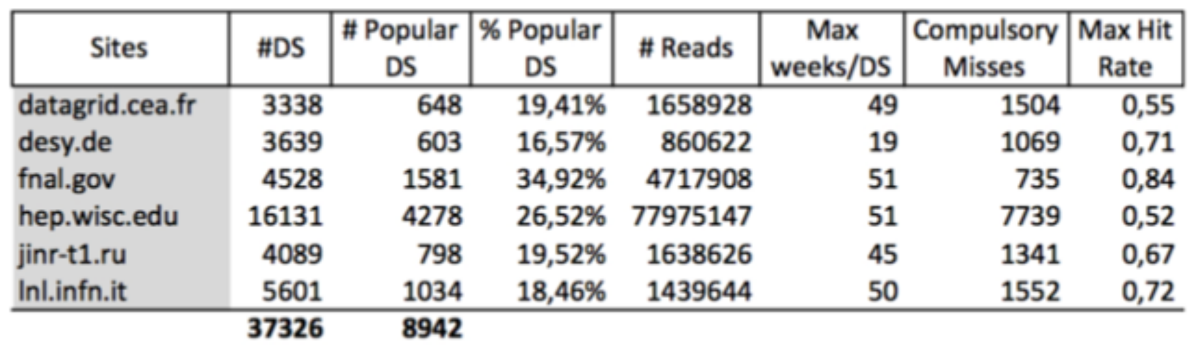
Caching strategies for sites

All this can be seeded to next generation of CMS Dynamic DM system, and lead to better site utilisation. But - regardless of what CMS centrally does - **can a site use this info today? Yes.**

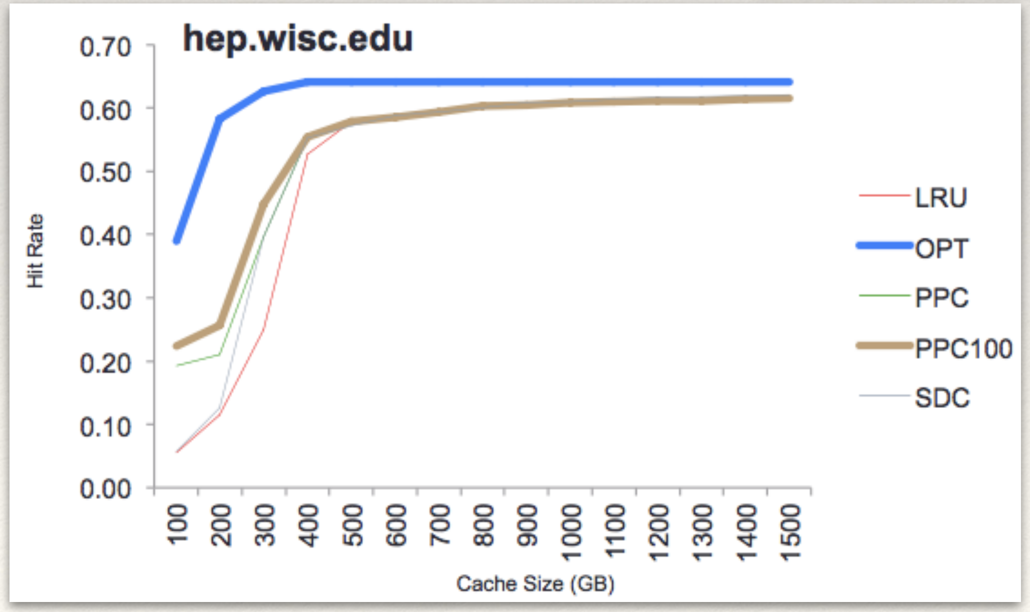
Apply popularity predictions to dataset placement on a site

- Strategy: do not evict cache elements if predicted to be popular for next N weeks

To-do: comparison between ML-based and human-based approaches by studying specific sites



Sites	#DS	# Popular DS	% Popular DS	# Reads	Max weeks/DS	Compulsory Misses	Max Hit Rate
datagrid.cea.fr	3338	648	19,41%	1658928	49	1504	0,55
desy.de	3639	603	16,57%	860622	19	1069	0,71
fnal.gov	4528	1581	34,92%	4717908	51	735	0,84
hep.wisc.edu	16131	4278	26,52%	77975147	51	7739	0,52
jinr-t1.ru	4089	798	19,52%	1638626	45	1341	0,67
lnl.infn.it	5601	1034	18,46%	1439644	50	1552	0,72
	37326	**8942**					



LRU Least Recently Used
OPT Bèlàdy's Algorithm
PPC Popular Prediction Caching
PPC100 w/100% accurate classifier
SDC Static Dynamic Cache
MaxHitRate = 1 - compulsoryMisses/nDS

Transfer latencies [*]

[classification + regression]

[*] In general, how long it takes to transfer different datasets across a complex topology of WLCG Tiers.

Goal: Transfer latencies impact differently the various analysis and production workflows: ML can help to classify then, and use predictions in the routing logic in any experiment-level DM tool.

Refs: [5-8]

Understanding latencies

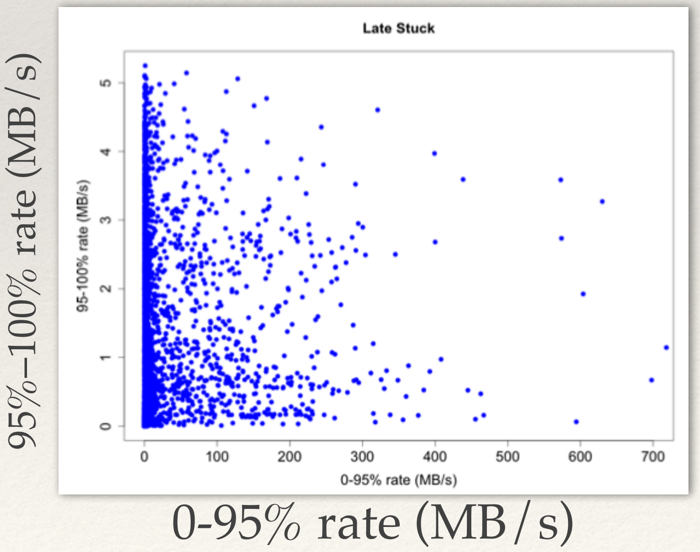
Base ingredients: logs from **PhEDEx** and **FTS**

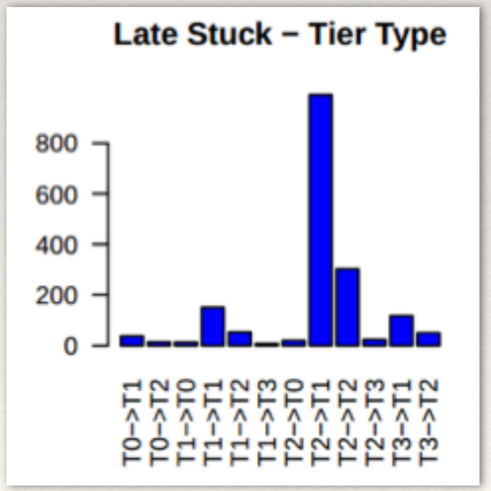
- **PhEDEx**: current CMS reliable and scalable dataset replication system
- **FTS**: File Transfer Service used by LHC experiments

Preliminary and preparatory work since long ago

- CMS PhEDEx equipped to save the relevant data indefinitely (*very important!*)
- Exploratory analysis on latency types and signatures, use of *skew variables* to describe them properly, etc.. Identification of transfer clusters/categories, e.g. *late* and *early* stuck

Example: *late stuck* may occur while transferring large datasets (due to e.g. transient storage issues or corrupted files). Identified cases well in advance may help Ops team to cure, and ultimately automatic procedures to be set up.





Latencies: data handling

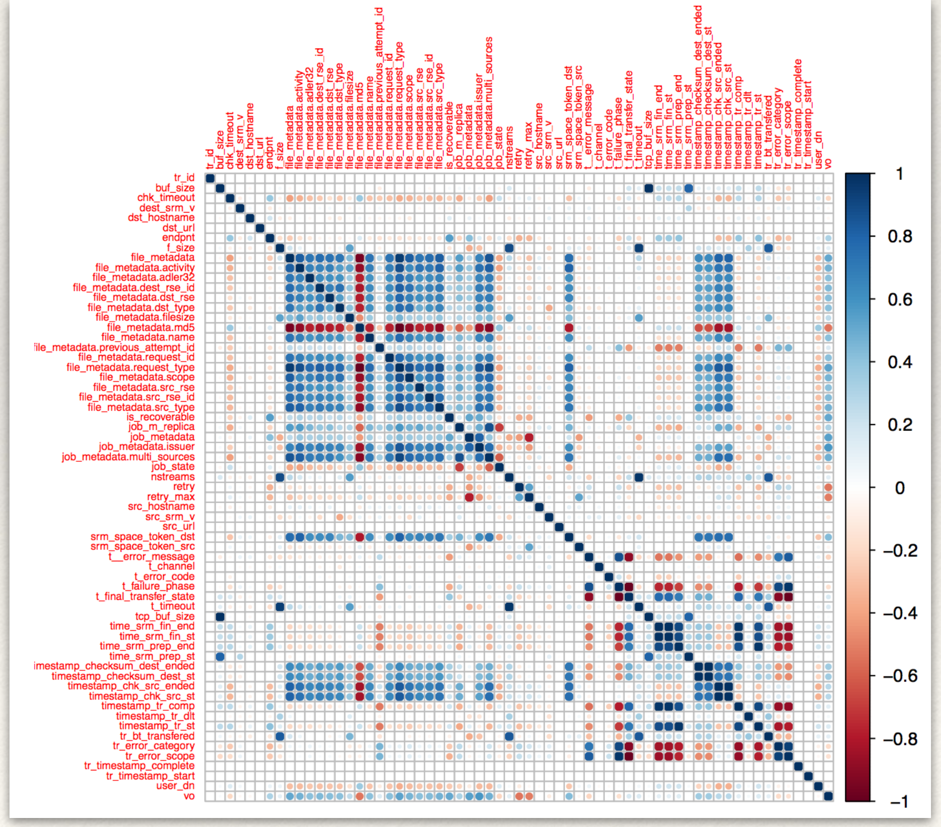
Transform PhEDEx/FTS data into a ML suitable form

- FTS $\mathcal{O}$(TB) raw data collected and injected into a CERN HDFS cluster
- convert JSON objects in ASCII files to a flat table format (CSV)
    - JSON doc structure loosely matched, custom EOF, etc
    - each record has nested records, need to be flattened
- hashing algorithms used to convert text → numerical values
- placeholder manually set for all missing attributes
- train on $\mathcal{O}$(GB) input files

Care in data preparation yields reduction to ~50% of the original CMS dataset

- attributes about the end of a (e.g. failure occurrences, transfer Δt) can't be used for predictions
- attributes that are static through all datasets are uninformative, waste space and even may mislead algorithms, so they should be dropped
- obviously correlated attributes (e.g. # files vs file size)

e.g. FTS logs



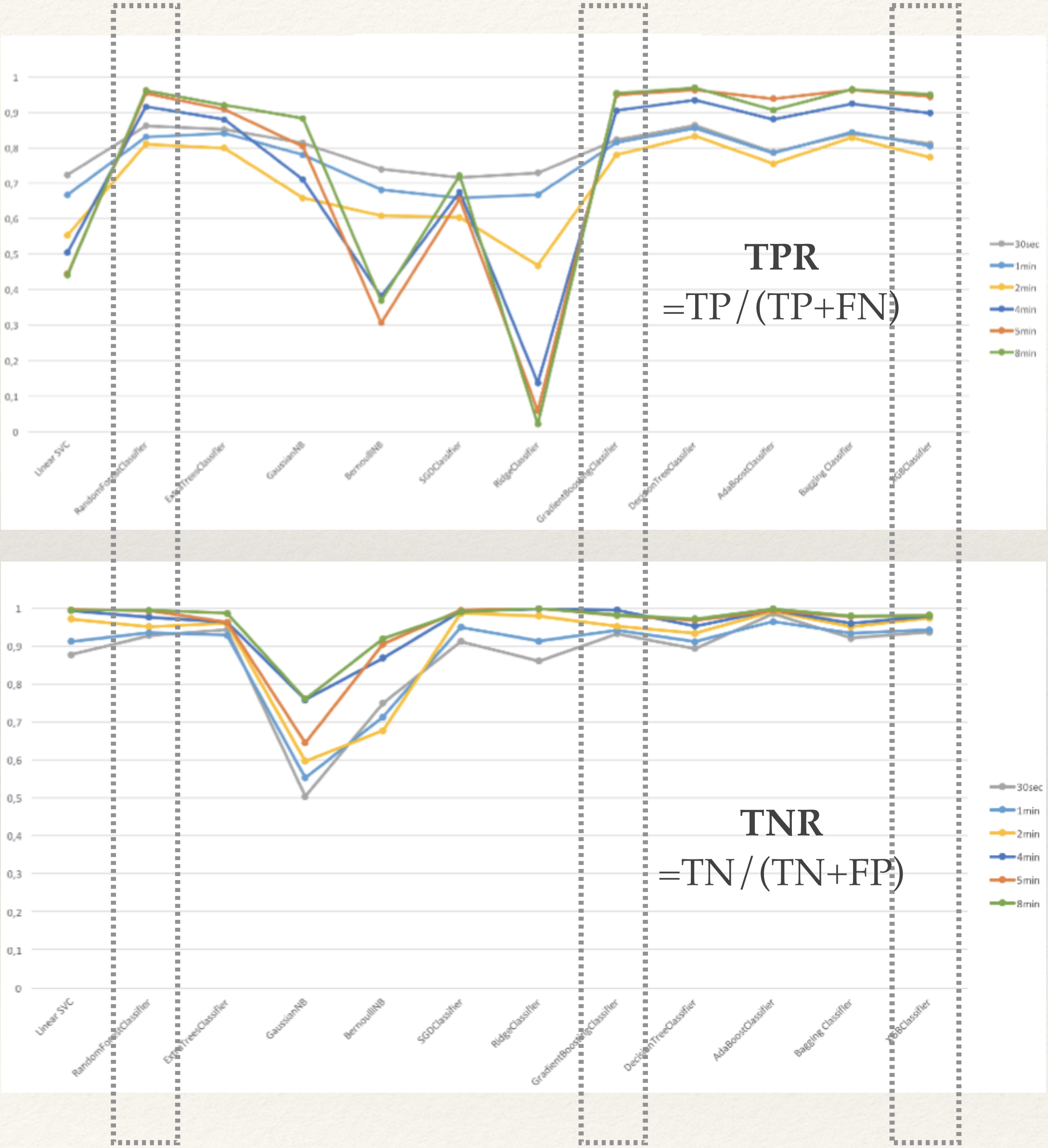
Latencies: a note on Spark

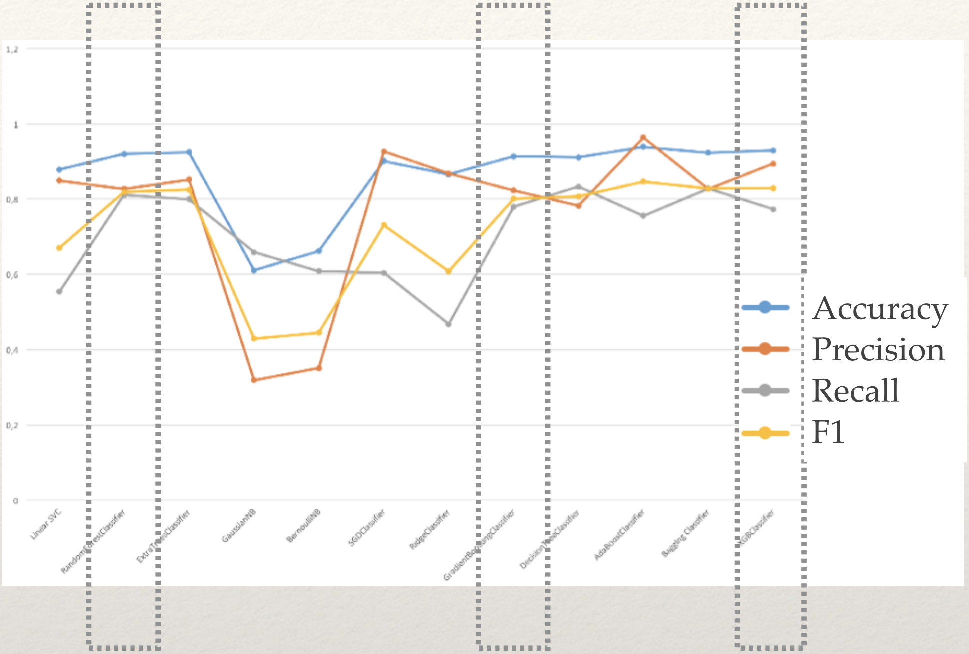
(Again): a **Spark** platform can solve many issues outlined above and provide cost-effective solutions to prepare data for ML box

- **CMSSpark** framework developed in CMS, used also here
- It can parse logs/data from a ~dozen of CMS (and not) data-services, e.g. AAA, EOS, FTS, CRAB, CMSSW, HTCondor, DBS, PhEDEx, WMAgent, etc.

Preliminary results

RandomForest GradientBoost XGB





Achieve ~**80%** accuracy, but work in progress still..

Tree-based classifiers among the best ones.

Focus on: **RandomForest Regressor** and **GradientBoost Regressor** from Scikit-learn lib, **XGBRegressor** from XGBoost lib - in terms of scorers (here) but also time and memory consumption (not shown here)

Possible applications

Deeper knowledge on transfer system behaviour with help from ML may help to properly tackle transfer congestions and thus reduce ETAs.

Empower the transfer system *router* decisions in future DM tools with such insight. A prototype:

- https://github.com/vkuznet/**transfer2go**
  - **R&D** with a Google Summer of Code student in Summer 2017)

More work ahead both in terms of verifying/improving ML predictions and code architecture to implement them into existing/ new transfer system.

Workflow handling [*]

[unsupervised]

[*] referring to the CMS Workflow Management sector, and specifically to the work done by the team(s) that manage centrally-orchestrated Grid/Cloud jobs in CMS

Goal: Deliver error handling predictions and mature towards moving manual operator intervention into automated actions.

Workflow handling needs

Central CMS MC production manages thousands of "workflows" each with thousands of jobs.

Common issues are errors in Grid jobs about missing/corrupt input files, high memory usage, etc.

All currently handled manually in Ops

- An operator must look at the error codes and decide actions to be taken
- BUT: 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 important - impossible to maintain

ML stands as a very natural solution

- Goal: **limit**, **group**, **enumerate** possible operator actions
  - for well defined groups of errors, operators do kill/clone/resubmit/recover as appropriate

Strategy and implementation

Put unsupervised ML <u>before</u> the operator

Pull in data to build a model from CMS data services

- currently: errors thrown (WMStats), site statuses (SiteDB and CMS Site Support team)
  - next: more workflow parameters (Request Manager), job splitting, XRootD enabled or not, requested memory, ..
- basically, a matrix: # times each error codes occurs per site, times site status at the time of the error (e.g. enabled, disabled, drain). Collapsing into *good* or *bad* site statuses to limit complexity to x2 only

Use **K-Means clustering**

- group similar error-type workflows together (multiple workflows can then be acted on at the same time)
- make the cluster characteristics stable by including the stored workflow error patterns and site statuses over the past few months
- in progress: techniques to compress errors and sites phase space in order to group similar errors better, and to fight against sparse matrices

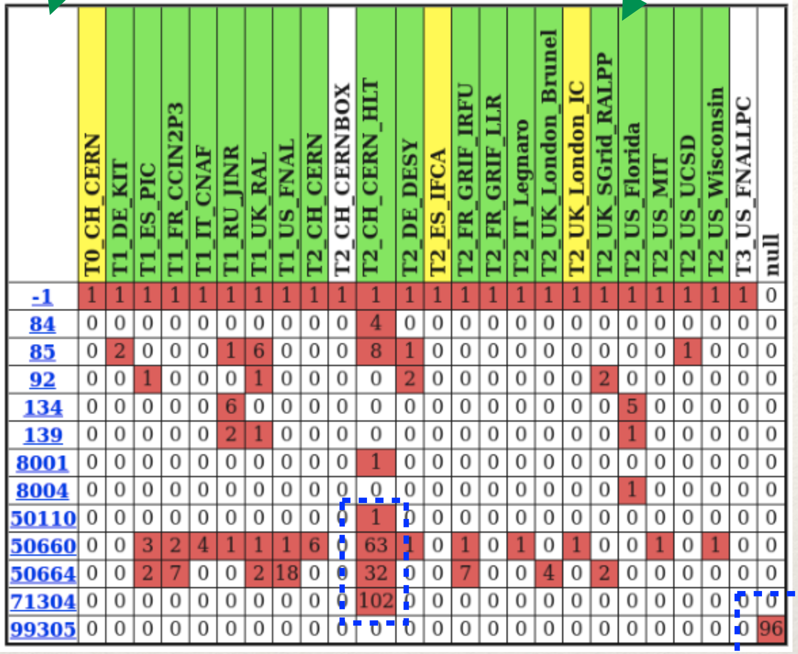
Python libs used:

- **Keras** for deep net modelling; **Scikit-learn** library to handle model building pipeline, applying feature scaling, resampling, and modelling; **Pandas** to store input data as data frames; **Matplotlib** for plotting; **Scipy** to handle computation such as KS-test, p-value

One of these tables per **EACH CMS WORKFLOW**

SITES
The color of each site corresponds to its status (white means the site status is not tracked by Dashboard, and it is assumed "good")

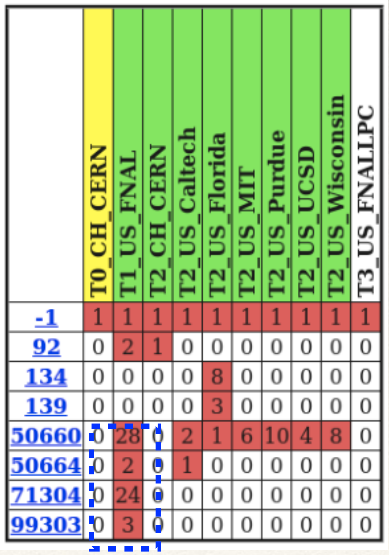
The **size** of the pie charts corresponds to the # errors. Each **color** in the pie chart is a different site.



	T0_CH_CERN	T1_DE_KIT	T1_ES_PIC	T1_FR_CCIN2P3	T1_IT_CNAF	T1_RU_JINR	T1_UK_RAL	T1_US_FNAL	T2_CH_CERN	T2_CH_CERNBOX	T2_CH_CERN_HLT	T2_DE_DESY	T2_ES_IFCA	T2_FR_GRIF_IRFU	T2_FR_GRIF_LLR	T2_IT_Legnaro	T2_UK_London_Brunel	T2_UK_London_IC	T2_UK_SGrid_RALPP	T2_US_Florida	T2_US_MIT	T2_US_UCSD	T2_US_Wisconsin	T3_US_FNALLPC	null
-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
84	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
85	0	2	0	0	0	1	6	0	0	0	8	1	0	0	0	0	0	0	0	0	0	1	0	0	0
92	0	0	1	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	2	0	0	0	0	0	0
134	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0
139	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
8001	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8004	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
50110	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
50660	0	0	3	2	4	1	1	1	6	0	63	1	0	1	0	1	0	1	0	0	1	0	1	0	0
50664	0	0	2	7	0	0	2	18	0	0	32	0	0	7	0	0	4	0	2	0	0	0	0	0	0
71304	0	0	0	0	0	0	0	0	0	0	102	0	0	0	0	0	0	0	0	0	0	0	0	0	0
99305	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96

ERROR CODES

A not-obviously similar workflow to the one above, but the same errors at different sites are likely from similar causes

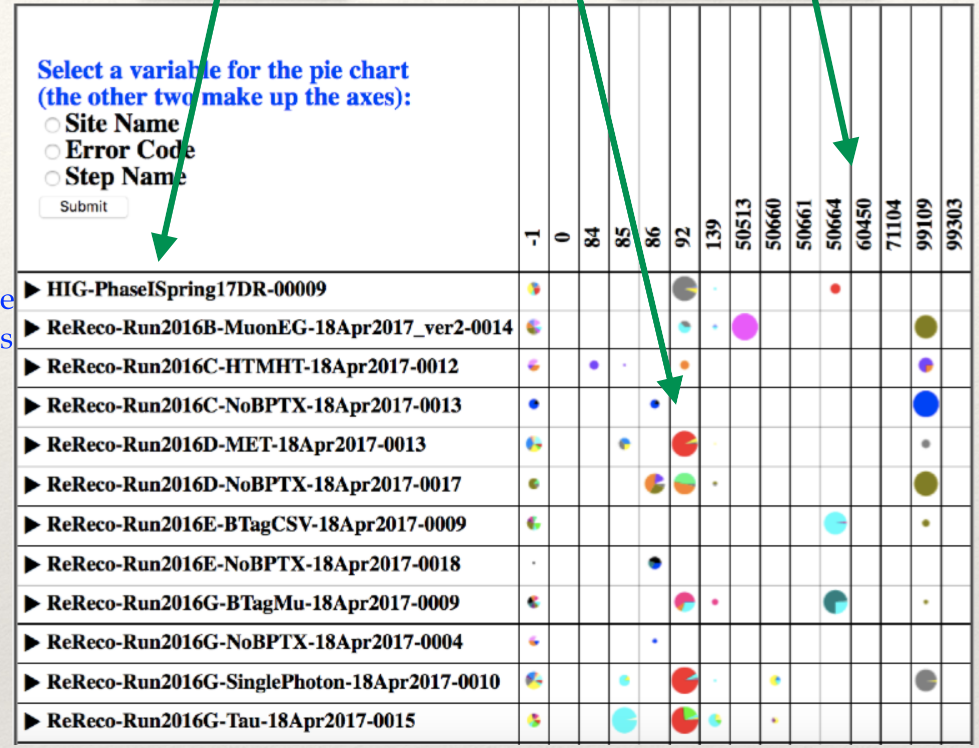


	T0_CH_CERN	T1_US_FNAL	T2_CH_CERN	T2_US_Caltech	T2_US_Florida	T2_US_MIT	T2_US_Purdue	T2_US_UCSD	T2_US_Wisconsin	T3_US_FNALLPC
-1	1	1	1	1	1	1	1	1	1	1
92	0	2	1	0	0	0	0	0	0	0
134	0	0	0	0	8	0	0	0	0	0
139	0	0	0	0	3	0	0	0	0	0
50660	0	28	0	2	1	6	10	4	8	0
50664	0	2	0	1	0	0	0	0	0	0
71304	0	24	0	0	0	0	0	0	0	0
99303	0	3	0	0	0	0	0	0	0	0

Workflows that are similar get clustered together. All features are dumped into a K-means clustering algorithm.

WORKFLOW

ERROR CODES



Select a variable for the pie chart (the other two make up the axes):

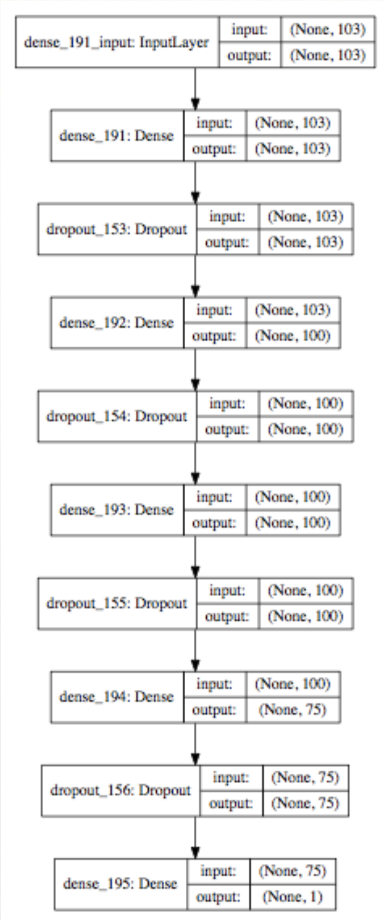
○ **Site Name**
○ **Error Code**
○ **Step Name**

- ▶ HIG-PhaseISpring17DR-00009
- ▶ ReReco-Run2016B-MuonEG-18Apr2017_ver2-0014
- ▶ ReReco-Run2016C-HTMHT-18Apr2017-0012
- ▶ ReReco-Run2016C-NoBPTX-18Apr2017-0013
- ▶ ReReco-Run2016D-MET-18Apr2017-0013
- ▶ ReReco-Run2016D-NoBPTX-18Apr2017-0017
- ▶ ReReco-Run2016E-BTagCSV-18Apr2017-0009
- ▶ ReReco-Run2016E-NoBPTX-18Apr2017-0018
- ▶ ReReco-Run2016G-BTagMu-18Apr2017-0009
- ▶ ReReco-Run2016G-NoBPTX-18Apr2017-0004
- ▶ ReReco-Run2016G-SinglePhoton-18Apr2017-0010
- ▶ ReReco-Run2016G-Tau-18Apr2017-0015

Prototype of interface for operator **built** and **in use**: capable of clustering workflows with similar errors

Now **recording the actions taken by operators** in a manner that can be trained.

Alternative approach with **deep neural network**



dense_191_input: InputLayer | input: (None, 103) | output: (None, 103)
dense_191: Dense | input: (None, 103) | output: (None, 103)
dropout_153: Dropout | input: (None, 103) | output: (None, 103)
dense_192: Dense | input: (None, 103) | output: (None, 100)
dropout_154: Dropout | input: (None, 100) | output: (None, 100)
dense_193: Dense | input: (None, 100) | output: (None, 100)
dropout_155: Dropout | input: (None, 100) | output: (None, 100)
dense_194: Dense | input: (None, 100) | output: (None, 75)
dropout_156: Dropout | input: (None, 75) | output: (None, 75)
dense_195: Dense | input: (None, 75) | output: (None, 1)

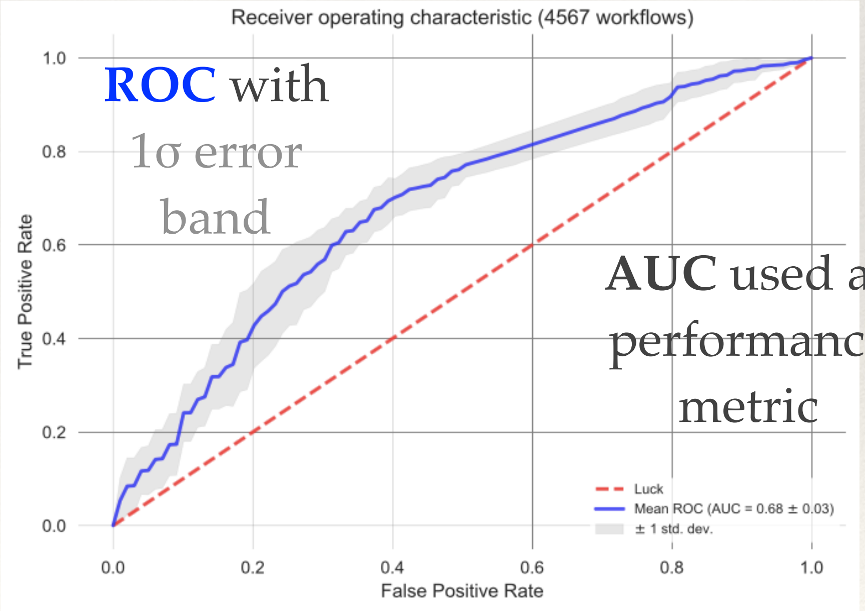
Deep net arch (not optimised yet):
- 5 Layers
- 75 neurons (hidden units) per layer
- RELU as activation function
- Dropout 0.002
- Learning rate 1e-3
- Binary cross-entropy as loss function
- ADAM for GD optimiser

An alternative **supervised** approach with **DNN** can be adopted

- <u>Goal</u>: predicting recover (*ACDC*) actions versus resubmit (*clone*) actions
- using the same "error codes - site status" matrix information as in previous slide (e.g. top left)

Deep net **implemented**

- *Preliminary*: at first attempt, already **(68±3)%** accurate (AUC)
- And it can be further optimised, cured for over-training, etc.: work in progress..



ROC with 1σ error band

AUC used as performance metric

Event classification [*]

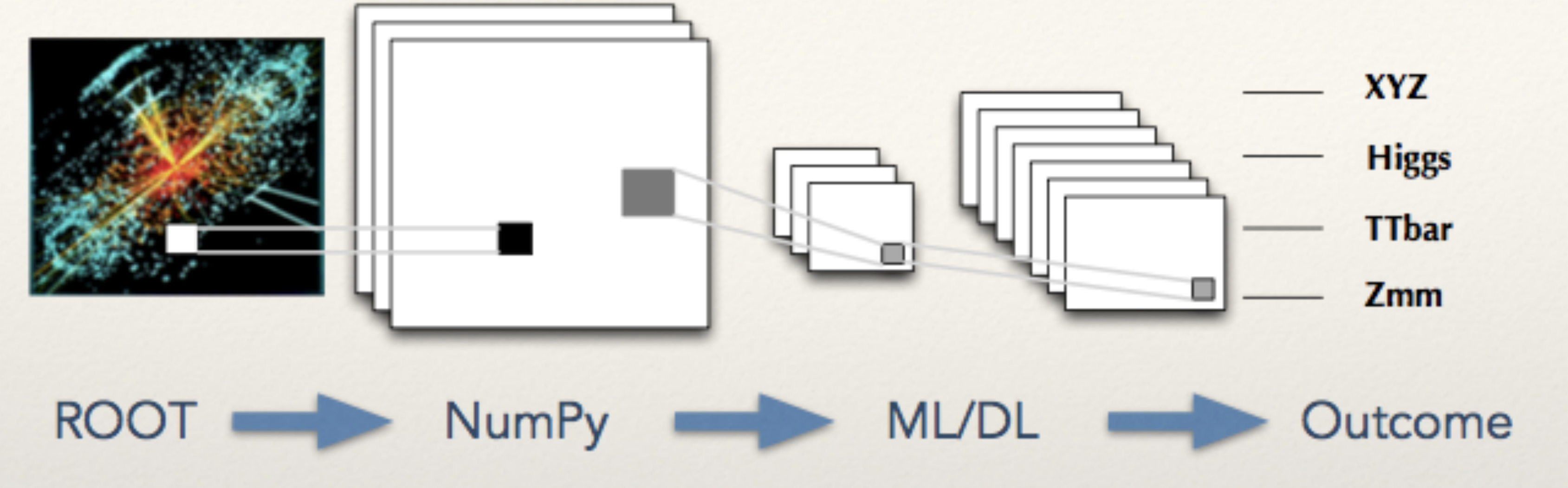
[unsupervised]

[*] Classify physics events in general.

Goal: Explore novel ways to do so within computing resources budget constraints.

Refs: [9-12]

Event classification problem

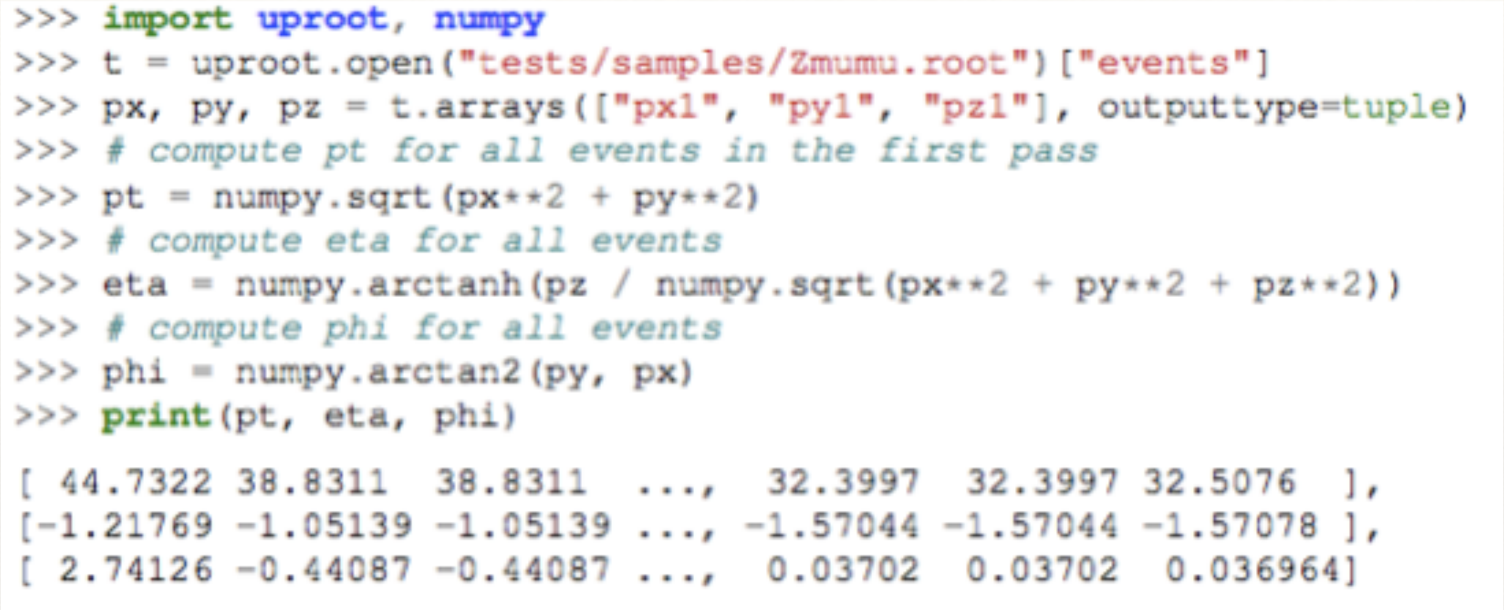


ROOT to NumPy transformation allows to use world-class ML/DL frameworks and tools for HEP data (focus not only CMS here)

ROOT from Python

DIANA-HEP "*uproot*" project

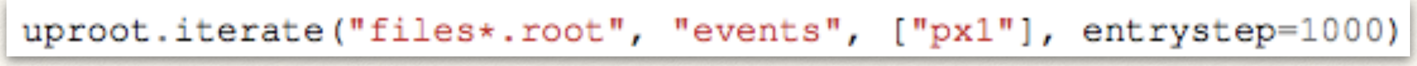
- https://github.com/scikit-hep/uproot
- pure Python ROOT reader that directly copies columnar ROOT data into NumPy arrays



```
>>> import uproot, numpy
>>> t = uproot.open("tests/samples/Zmumu.root")["events"]
>>> px, py, pz = t.arrays(["px1", "py1", "pz1"], outputtype=tuple)
>>> # compute pt for all events in the first pass
>>> pt = numpy.sqrt(px**2 + py**2)
>>> # compute eta for all events
>>> eta = numpy.arctanh(pz / numpy.sqrt(px**2 + py**2 + pz**2))
>>> # compute phi for all events
>>> phi = numpy.arctan2(py, px)
>>> print(pt, eta, phi)

[ 44.7322 38.8311   38.8311  ...,   32.3997   32.3997 32.5076  ],
[-1.21769 -1.05139 -1.05139 ..., -1.57044 -1.57044 -1.57078 ],
[ 2.74126 -0.44087 -0.44087 ...,   0.03702   0.03702   0.036964]
```

E.g. a loop over events is not in (slow) Python for loop, but in (fast) compiled code with NumPy arrays behind (and iteration in batches foreseen for big datasets)

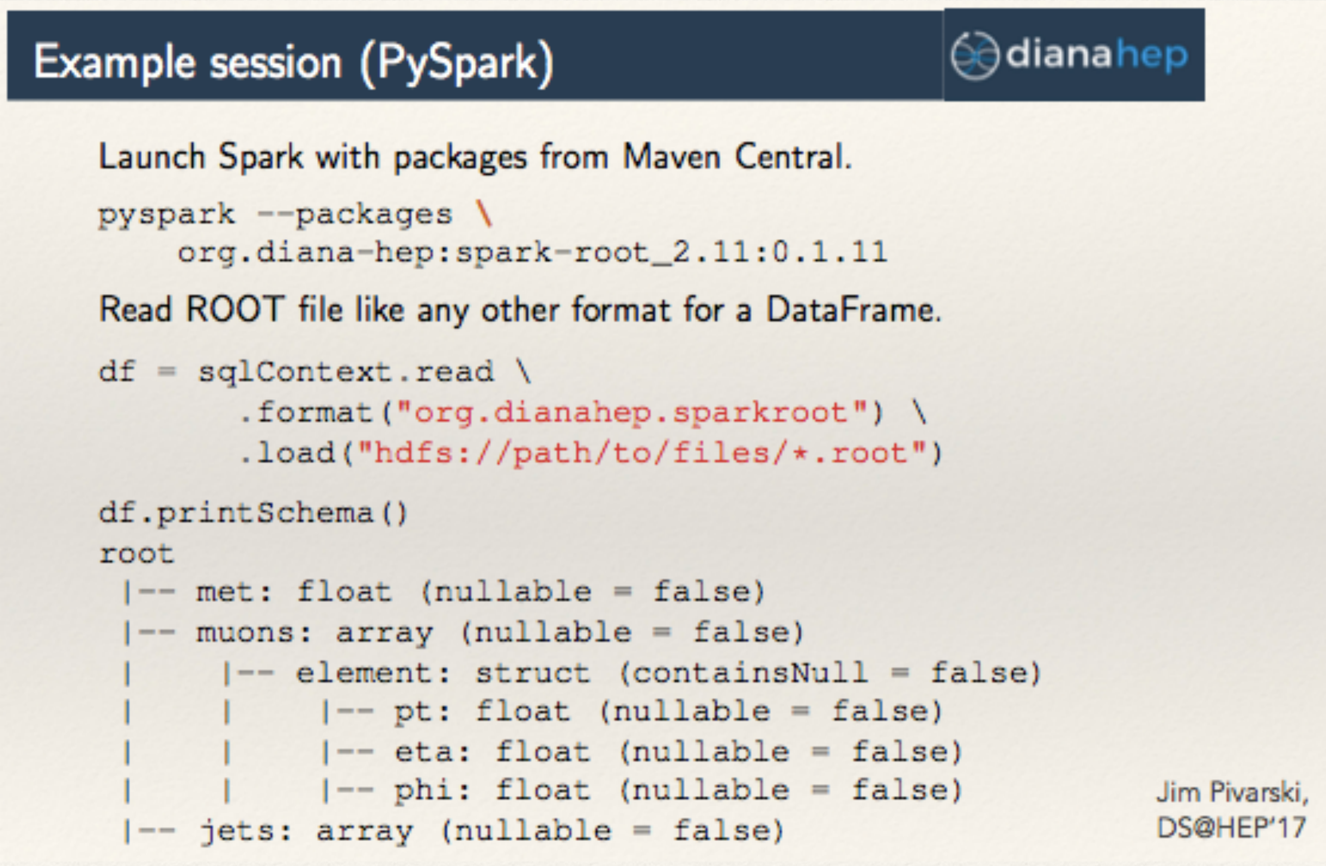


```
uproot.iterate("files*.root", "events", ["px1"], entrystep=1000)
```

Dealing with large datasets

For big datasets, things get better with Spark

- now possible to read in ROOT files on Spark platform natively



Example session (PySpark)

Launch Spark with packages from Maven Central.

```
pyspark --packages \
    org.diana-hep:spark-root_2.11:0.1.11
```

Read ROOT file like any other format for a DataFrame.

```
df = sqlContext.read \
       .format("org.dianahep.sparkroot") \
       .load("hdfs://path/to/files/*.root")

df.printSchema()
root
 |-- met: float (nullable = false)
 |-- muons: array (nullable = false)
 |    |-- element: struct (containsNull = false)
 |    |    |-- pt: float (nullable = false)
 |    |    |-- eta: float (nullable = false)
 |    |    |-- phi: float (nullable = false)
 |-- jets: array (nullable = false)
```

Jim Pivarski,
DS@HEP'17

ML/DL as a service

Performing ML/DL in production at scale is a challenge

- pushing a ML-based project beyond a basic exploratory phase is not trivial
  - reading ROOT files efficiently, use world-class ML/DL models, profit of adequately scaled infrastructure from training, data-science skills needed (sometimes complementary to physics analysis)

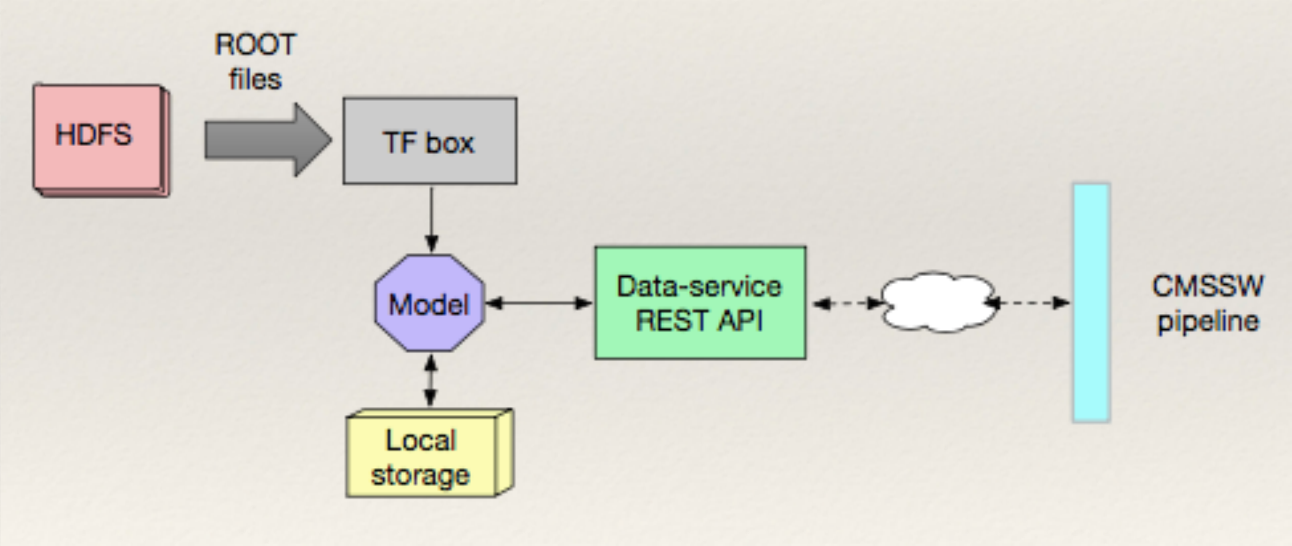
Data transformation steps, and streaming to emerging frameworks (e.g. Google Tensorflow) as well as allowing access to solid hardware resources for training (e.g. GPUs, HDFS cluster, ..) can be designed "as a service" for many use-cases

More in next slides.

Google Tensorflow as a service for CMS (**TFaaS**)

An end-to-end data-service able to

- read ROOT and convert it into ML/DL input (e.g. via *uproot*)
- serve ML/DL models via REST API
  - data exchange via high efficient transport layer (e.g. proto-buffers)
  - read data remotely and integrate service calls into CMSSW
- ability to deploy the service to the cloud (e.g. rent GPUs to train model)



Prototype ready. Proof of concept done for 2 use-cases:

1) event classification; 2) S/B discrimination in all-hadronic top decays in CMS

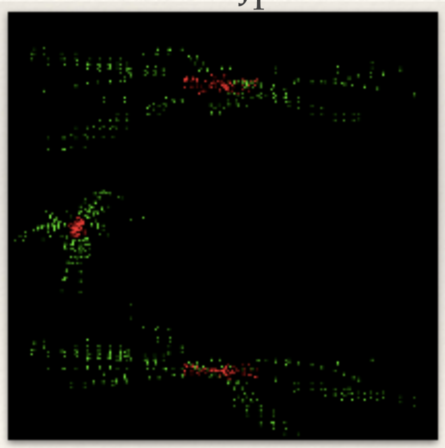
Docker image also available: https://hub.docker.com/r/veknet/tfaas/ (documentation is progress)

Event classification

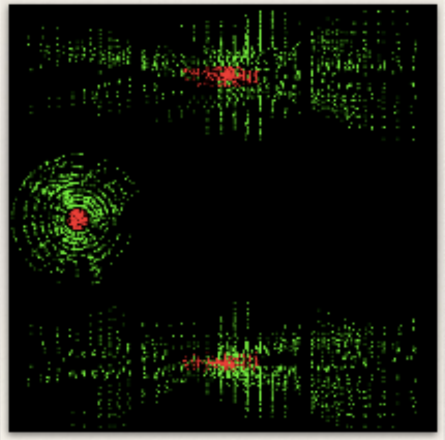
Application of image classification (e.g. CNN) to classify HEP events (i.e. no traditional tracking algos)

- extract Pixel/Silicon hits in global coordinate frame
- transform them into PNG images (single CMSSW EDAnalyzer)
- feed them to CNN; train sample: 27K images; CNN training using world-class models (http://www.fast.ai framework based on PyTorch)

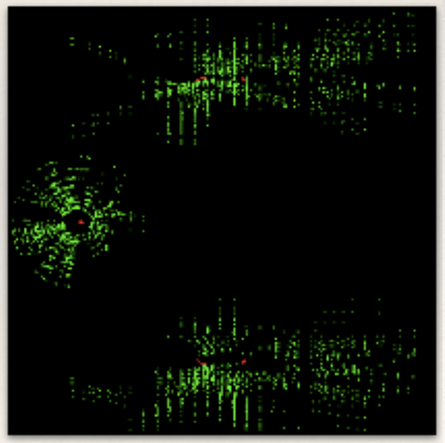
JpsiMuMu

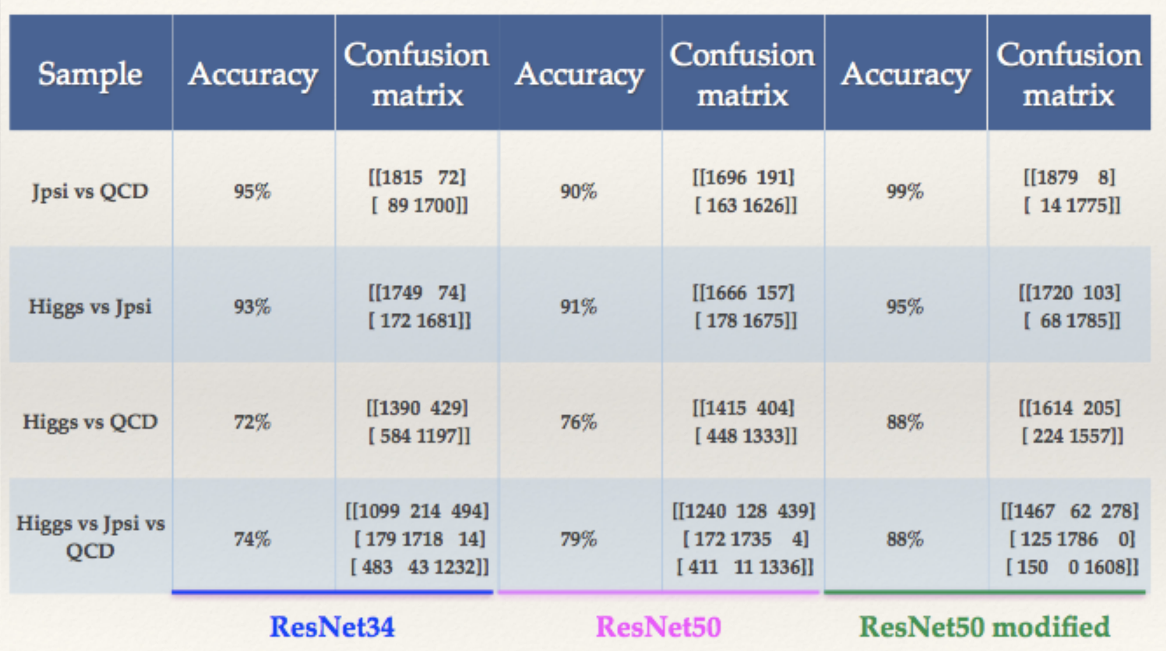


QCD



Higgs 200 Taus





Sample	Accuracy	Confusion matrix	Accuracy	Confusion matrix	Accuracy	Confusion matrix
Jpsi vs QCD	95%	[[1815 72] [89 1700]]	90%	[[1696 191] [163 1626]]	99%	[[1879 8] [14 1775]]
Higgs vs Jpsi	93%	[[1749 74] [172 1681]]	91%	[[1666 157] [178 1675]]	95%	[[1720 103] [68 1785]]
Higgs vs QCD	72%	[[1390 429] [584 1197]]	76%	[[1415 404] [448 1333]]	88%	[[1614 205] [224 1557]]
Higgs vs Jpsi vs QCD	74%	[[1099 214 494] [179 1718 14] [483 43 1232]]	79%	[[1240 128 439] [172 1735 4] [411 11 1336]]	88%	[[1467 62 278] [125 1786 0] [150 0 1608]]

ResNet34 ResNet50 ResNet50 modified

TFaaS demo: from installation to operation (*for S/B in top quark analysis)*

https://drive.google.com/open?id=1g-_23rgEB4T0vGrScAwdy33PSjOZ7TAT)

Summary

Exploration of ML/DL for CMS computing based on existing (meta)data is advancing in various areas

- experience on dataset popularity, transfer latencies, job resubmission tactics
- event classification and a physics analysis use-cases (S/B in all hadronic top) used to understand how to build ML/DL ``as a service'' for CMS
- more projects open and eventually gaining boost, too

A note on manpower

- remarkably, most of the work is done by **students** interested in applying data science practices to the HEP domain

Values (in random order..)

- approaches designed to be pluggable into any (current or future) DM/WM system
- proof-of-concepts of ML/DL to be used for CMS Computing model evolutions
- efforts towards making ML/DL accessible by *anyone* in CMS
- in few cases (e.g. FTS), synergy with other communities can be envisioned

References

[1] *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

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

[3] *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

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

[5] *No file left behind - monitoring transfer latencies in PhEDEx*, proceedings of CHEP 2012, J. Phys.: Conf. Ser. 396 032089, http://iopscience.iop.org/article/10.1088/1742-6596/396/3/032089/pdf

[6] *Monitoring data transfer latency in CMS computing operations*, proceedings of CHEP 2015, http://stacks.iop.org/1742-6596/664/i=3/a=032033

[7] CERN Summer student report, https://cds.cern.ch/record/2209068/files/CERN_REPORT_Z_MATONIS%20docx%20(1).pdf

[8] CERN Summer Student report, https://cds.cern.ch/record/2218016/files/TommasoDiotalevi_report.pdf

[9] Middleware solution to convert ROOT to NumPy, https://github.com/diana-hep/c2numpy

[10] Middleware solution to read ROOT natively in Python: https://github.com/scikit-hep/uproot

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

[12] *Prototype of ML as a service for CMS Physics in S vs B discrimination*, L. Giommi, Master thesis (dissertation soon after ISGC'18)