attilio.santocchia@gmail.com

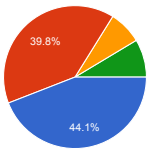
Edit this form

93 responses

View all responses Publish analytics

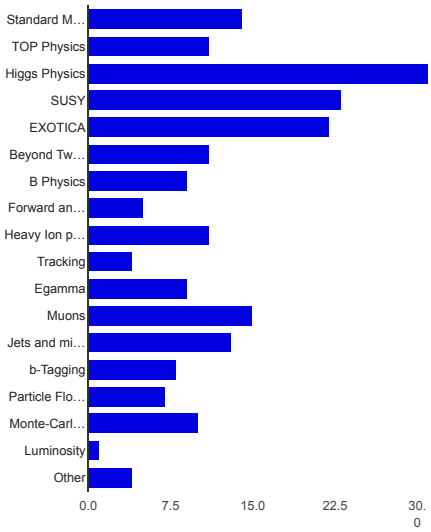
Summary

Define yourself



Analysis User (you don't write code for cmssw or/and for any other framework, you write code for your own analysis)	41	44.1%
Developer User (you write code for a framework used by your group to produce intermediate root trees for your analysis)	37	39.8%
Core Developer User (you write and commit code for CMSSW)	7	7.5%
Other	8	8.6%

POG or PAG of interest



Standard Model Physics	14	15.1%
TOP Physics	11	11.8%
Higgs Physics	31	33.3%
SUSY	23	24.7%
EXOTICA	22	23.7%
Beyond Two Generations	11	11.8%
B Physics	9	9.7%
Forward and Small-x QCD Physics	5	5.4%
Heavy Ion physics	11	11.8%
Tracking	4	4.3%
Egamma	9	9.7%
Muons	15	16.1%
Jets and missing energy (JetMET)	13	14%
b-Tagging	8	8.6%
Particle Flow and tau-id	7	7.5%
Monte-Carlo generators and physics utilities	10	10.8%
Luminosity	1	1.1%
Other	4	4.3%

If you are working on one or more analysis, could you tell us the CADI code?

HIG-16-043
HIG-16-022
HIG-16-036 SMP-16-002
EXO-16-022
HIG-16-035
HIG-16-021, HIG-16-023, SMP-16-006
No CADI lines defined yet...
SUS-16-033
SUS-16-017, SUS-16-045
EXO-16-037
Mostly working as MC contact
FSQ-15-006, FSQ-16-002, FSQ-16-003
SMP-16-015, SMP-16-005
PRF-14-001
HIG-16-038
FSQ-13-007
TOP-16-010
B2G-16-021, B2G-16-004, B2G-16-007
SMP-15-008
HIG-16-011 ; HIG-16-024
HIG-16-015
EXO-16-003, EXO-16-041, SUS-16-003
SUS-16-039
HIG-14-040
TOP-16-006, TOP-16-019, TOP-16-023

HIG-16-040
TOP-16-004 (in the past)
HIG-15-013
EXO-16-048
HIN-15-014
HIN-16-010
BPH-16-002, BPH-16-003
HIN-13-005, HIN-14-001, HIN-14-007, And two new analysis with HIN and one with SMP-J
HIN-16-004
HIN-16-015
B2G-16-020, B2G-16-029, SMP-14-018
B2G-17-001
B2G-15-008
HIN-16-014
B2G-16-017
HIN-16-006, HIN-16-024, TOP-16-015, TOP-16-023, JME-16-002
B2G-16-022, B2G-17-005
BPH-15-001, BPH-15-005
B2G-16-015, SMP-16-015, others
HIG-16-044
B2G-16-024, upcoming B2G and HIG cadi lines
TOP-16-019
HIG-16-017
B2G-15-006; B2G-16-019
HIG-16-002
EXO-16-037,EXO-16-038,EXO-16-039,EXO-16-040,EXO-16-048,EXO-16-050,EXO-16-051,EXO-16-052,EXO-16-053
SUS-16-035, SUS-16-036
EXO-16-052
EXO-16-057
too many to list
SUS-16-035
HIN-13-005
AN-16-456, AN-16-461
No CADI lines assigned yet
SMP-14-014, SMP-15-005, SMP-16-006, SMP-16-001, SMP-16-002, SMP-16-017, SMP-16-019
SUS-16-034, SUS-16-036, SUS-16-043
BPH-14-003
BPH-15-001
HIN-15-003 (completed). Two BPH analyses will (hopefully) be in CADI soon.

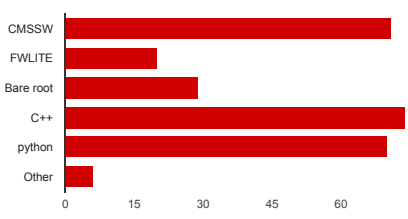
Source of your analysis (before any further reduction/processing... as extracted from DAS)

MiniAOD	**69**	74.2%
AOD	**27**	29%
RECO	**11**	11.8%

If you selected more than one data tiar (i.e. AOD and MiniAOD) in the previous section, could you explain why?

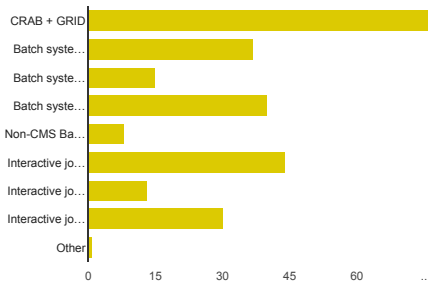
For very special data (Run2015A 0T) we need RECO for special re-reconstruction on the fly. Standard tracking for example is not good enough.
For detailed reconstruction studies, necessary products not available in mini AOD (understandably)
at times need RAW to rerun HLT
miniAOD: suitable for jet&DPS analyses / standard high pile-up data; RECO: needed for work with basic low pt/low energy objects in low pile-up data (cross section analyses, energy flow, UE etc...).
For one analysis, we need calojets and general tracks, which are only available in miniaod
Some of the heavy ion objects such as centrality were not there in miniAOD or rechits etc...
for vertex reconstruction in B decays, miniAOD does not have full error information on tracks
AOD is smaller than RECO, but some data can only be found in RECO tier
Most of the analysis is on MiniAOD, but muon POG TnP trees run on AOD
We perform fits in wide timing window to measure rate of spike and beam halo events in the monophoton analysis. Intend to use Skims for that in future.
Traditionally using AOD, studies are ongoing to check if miniAOD can fit
We use AOD the for reconstruction of long lived particles. I use MiniAOD in more standard analyses.

What kind of environment/language do you use for your analysis (please select one or more item if needed)?



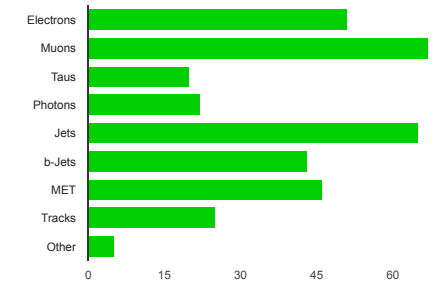
CMSSW	**71**	77.2%
FWLITE	**20**	21.7%
Bare root	**29**	31.5%
C++	**74**	80.4%
python	**70**	76.1%
Other	**6**	6.5%

What kind of computing infrastructure do you use?



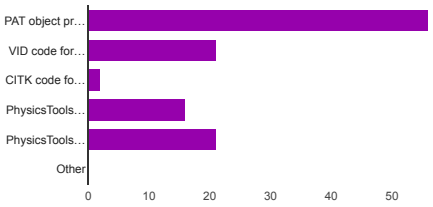
CRAB + GRID	**76**	81.7%
Batch system @ CERN	**37**	39.8%
Batch system @ FNAL	**15**	16.1%
Batch system @ T2/T3 (Not CERN or FNAL)	**40**	43%
Non-CMS Batch system (None of the previous options: the system is not part of the CMS computing infrastructure)	**8**	8.6%
Interactive jobs in a CMS site (T2 or T3)	**44**	47.3%
Interactive jobs in a non-CMS site	**13**	14%
Interactive jobs on my personal machine (laptop and/or desktop)	**30**	32.3%
Other	**1**	1.1%

What analysis objects are you mainly interested in?



Electrons	**51**	54.8%
Muons	**67**	72%
Taus	**20**	21.5%
Photons	**22**	23.7%
Jets	**65**	69.9%
b-Jets	**43**	46.2%
MET	**46**	49.5%
Tracks	**25**	26.9%
Other	**5**	5.4%

Are you using one of the following package included in CMSSW?



PAT object produced running on AOD/minAOD	**56**	77.8%
VID code for E/gamma IDs	**21**	29.2%
CITK code for isolation	**2**	2.8%
PhysicsTools/Heppy	**16**	22.2%
PhysicsTools/TagAndProbe	**21**	29.2%
Other	**0**	0%

Could you give us a simple description of the workflow that you normally use to produce final plots?

MiniAOD -> intermediate ntuple skims -> analysis specific ntuple -> plots

We skim miniAOD and use it directly to produce plots. We don't another intermediate data format.

MiniAOD or AOD -> flat ROOT tree with preselection -> histogram maker -> plotter

AOD -> PAT -> Ntuple -> Root/c++ -> Python -> Plots

MiniAOD->Ntuplizer->Analyzer->Plotter

miniAOD -> ROOT trees -> massaging on ROOT trees -> datacards & plots

Produce flat tuples with Heppy; run plotting/datacard creation on top.

miniAOD samples -> flat ntuples (branches for gen info, jets, MET, leptons, triggers, derived quantities, etc.) -> skimmed ntuples (apply baseline selections, use or remove systematic variation branches) -> histograms

MiniAOD -> "Big" ntuples shared by all members of the group -> "small" ntuples used for each individual analysis / analysis step -> simple root macros / python macros to produce final plots

run CRAB jobs to make ntuples from miniAOD and then run several ROOT macros, depending on which plots I want to make

Process miniAODs into slimmed (but not flat) ntuples containing only the objects, and variables, that we might be interested in. Run off these ntuples to produce flat trees used for plotting/making inputs to statistical tools

miniAOD processed in CMSSW to flat ntuples. Ntuples further skimmed per final state. Run a standalone very computing intense algo[*] which gives a most likely reconstructed mass for a Higgs (2 taus + met as input). This runs ~2 seconds per event and is by far the bottle neck of the analysis. After this, analysts further process data/make data cards. Then we use the Higgs Combine tool to make our final plots/limits. [*] https://github.com/veelken/SVfit_standalone

Produce ntuples from MiniAOD, run main analysis code on ntuples and write out smaller ntuples, produce plots from last set of ntuples with python modules.

Performance, Efficiencies, MC/data validation, reco-level plots, hadron-level plots

miniaod -> flat ntuple -> skimmed flat ntuple --(analysis step1)-> intermediate histograms --(analysis step 2)-> final histograms and figures

run heppy on miniaod to get flat ntuples (flat=no further loop necessary within one event). use python macros to read these flat ntuples with pyroot (TTree.Draw or TTree.Project) to make plots.

I create flat ntuples from regular miniAOD or enriched miniAOD (with additional trigger collections produced from RAW) using the cmgtools framework. I plot directly from these flat ntuples using again cmgtools code.

We produce trees using Heppy running on MiniAOD. We run our own framework (Python) to run on the trees, using modules like skimmers, producers and analyzers to produce the main results (yields to put in datacards, systematic studies, etc.). We use Combine to produce statistical results and simple macros in pyroot to produce plots.

Private multipurpose flat tree reader/analyzer

heppy ntuples 1x/mo on grid -> custom ntuples via a heppy looper 1x/week on grid/T3 batch -> plots/histograms via PyROOT 1x/day on T3 batch -> analysis of plots in jupyter notebook on T3 interactive/laptop

crab->nTuple-> nTuples analysis on lxplus-> unfold on my computer

Produce reduced trees from MINIAOD, develop analysis on them, produce additional information (scale factors, mva outputs, fake ratios) in friend trees, produce plots from the reduced trees and friend trees.

MiniAOD-Cattuples(oriented to our group)-MiniTree (for the analysis)-Plots, yield, etc

run batch jobs on MiniAOD applying corrections and SFs, run script for plots and combine

dataset (RECO/miniAOD) --> (crab3) --> root ntuple (saved on T2 storage) --> Proof python analyser producing histograms (multi-core or through batch/PoD) --> python script to plot results.

1: produce flat trees with Heppy+CMGTools. 2: loop over the event data in the trees and summarizes them into multi-dimensional categorical data with AlphaTwirl (https://indico.cern.ch/event/590196/contributions/2380011/). 3: import the categorical data as data frames in R or pandas and analyze them. 4: make plots with R graphical packages Lattice or ggplot2

miniAOD->group nTuples->analysis framework->TMVA->Higgs Combine Tool

miniAOD -> framework (non-flat) trees -> histos / flat trees (MVA training)

First, I skim the data using a CMSSW cfg file that both reduces the EDM event size via "keep/drop" statements and makes basic selections to reduce the total number of events. Skims are stored on CERN EOS personal and PAG space. Next, I run CMSSW selectors and and analyzers to produce the plots of interest for each dataset. Finally, I run a ROOT script to combine the plots for each dataset into stacks, perform fits, etc.

We run our own analyzer over the AOD/miniAOD that produces ntuples in our own format that allows us to iterate quickly on selections. We then produce another tree of variables that we read into our plotting/fitting framework, which produces plots, tables, and datacards.

miniaod -> NTuple -> histo

ntuplizer (from MiniAOD to flat trees), then TFriendTrees to add additional variables and selections, later TDraw to do selections, plots, yields, datacards for HiggsCombine, etc. (for all steps using ROOT, python and C++)

mostly python + ROOT

MINIAOD -> TTree -> control plots + flat TTree -> final plots

miniAOD -> private ntuples/skims -> histograms/root trees after full event selection -> train BDT -> apply BDT -> produce output plots

MiniAOD --> Smaller specialized EDM --> RooFit Workspaces or ntuples --> C++/RooFit code and/or python scripts --> Higgs Combine Tool

crab jobs on miniaod produce own analysis framework file (not root based) -> selection/event reconstruction/mva training on local T2 cluster & produce final simple root tuples -> final statistical evaluation/plotting

Data Cards processed using the Higgs Combine tool for limit calculation. Then use the standard plotter with the output to make the limit plots and the pull plots for the nuisance parameters.

I produce trees by submitting CRAB jobs using our CMSSW-based analysis code. Then I skim the output trees using a simple ROOT macro interactively on UIs. The final step is to run a simple ROOT macro on the skimmed trees.

DT muon calibration workflow to produce calibration constants

Produce analysis level ntuples from official RECO/AOD. Skim the ntuples. Run histogrammer on the skimmed ntuples. run final plotter on histograms.

AOD -> ROOT tree -> RooDataSet -> do fits and store the results to ROOT files -> combine all to plots

After getting Ntuples from crab jobs on MiniAOD, analysing them and making plots with python/root and C++.

miniAOD -> group-internal NTuple -> analysis pre-selection -> analysis event interpretation and flat root trees -> single histograms -> combination of histograms (stacking, etc) -> polishing of selected plots for the publication

Run on AOD to produce skimmed TTree's via CRAB, run on skimmed TTree's with more event and object selections to produce single skimmed Tree via condor, run on skimmed TTree interactively to produce histogram, run on histograms to make final plots

Ntuplization using FWLite run on FNAL condor system, then C++ to process ntuples submitted to FNAL condor system. Tag and probe work done with CRAB

create ntuples from AOD -> skim ntuples -> pass through private analysis framework -> spits out histograms (result, systematic uncertainties, etc) -> make pretty plots with simple macros

MiniAOD -> [CMSSW/C++ framework] -> Ntuples -> [C++/python scripts] -> Skim -> [python analysis code] -> plots

AOD(miniAOD) -> ntuple -> C++ in root -> root

We are presently running a small private TTree maker directly off MINIAOD.

Skim the Heppy ntuples, configure the analysis tools, launch plotting scripts written in Python which use PyRoot.

FWlite framework to process MiniAOD into flat trees, C++/ROOT macros to further skim/process trees, python framework to draw histograms from trees including systematic uncertainties and produce plots.

generation privately to lhe files, conversion to miniaod, then to ntuples

miniAOD->Bacon->RootFiles->Plots

Run analysis using modules on CMSSW usually using CRAB and then take the ntuples generated and use ROOT to make final plots.

I make nTuples from MiniAOD via framework lite setup (normally made using Fermilab condor/xrootd). From these nTuples I produce root TTrees via analysis framework implemented in simple root (interactive jobs). From these I make plots (also interactive).

C++ to go from MiniAOD -> complex ntuples, python with FWlite for reduction to tree, python to compose plots from trees

MINIAOD->ntuples->Plots

The miniAOD is unpacked to a flat ntuple (A) that is used by ~20 people in my group. This ntuple (A) is slightly larger but much faster to run over than MiniAOD because of the compression. Then each analysis groups makes a small ntuple (B) from the common ntuple (A). The small ntuple (B) can be looped over interactively in ~20 minutes to obtain the full results of the analysis as histograms.

MINIAOD -> CRAB jobs -> 1kb/evt nTuples (with >1 lepton skim) -> TSelector (compiled in CMSSW, run interactive/proof/condor) -> Single root file, many plots/datasets -> Plotter/datacard creation (python)

read AOD, produce ntuples, eventually skim ntuples, produce plots

AOD > group ntuple ("skim") > histograms/final ntuples for signal and control regions > limit setting/plots

One (two) layers of TTrees after MINIAOD (AOD)

I use a group developed analyzer to analyze miniAOD samples into flat ntuples and then produce plots.

MiniAOD -> private format -> private baby format -> plots and tables for AN and paper

MINIAOD is run on with CRAB to produce intermediate flat ntuples suitable for several analyses. These ntuples are then run on by several analysis in our group to make even smaller ntuples suitable for their specific analysis using condor at UCSD T2. The latter ntuples are then the input to C++ scripts which are used to produce final plots/tables.

Tree/n-tuple production in CMSSW full FW, event selection etc applied in C Macros, final plots produced with C macros

1) ReReRECO (due to an error that had been made in a major HI ReRECO that affected this analysis of Data and MC including hiForest (ntuple) production, once that's done we 2) iterate on hiForest (interactive), occasionally making much smaller root-tree (ntuple) to make and re-make final plots on (on T2/T3/colleagues used laptop).

From miniaod we produce ntuples with c code through crab. Than we run c code on condor to make lots of histograms. Than to combine the plots and make the final ones python and c code is utilized the locally component.

MiniAOD --> group flat Ntuples --> ROOT --> plots

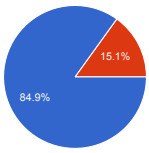
First we run on MiniAOD to make intermediate, flat root ntuples shared among multiple analyses. We then run (multiple times) on the intermediate ntuples to make smaller, analysis-specific ntuples for each analysis. The smaller ntuples are used to make plots and results.

I run one or more ROOT macro on the ROOT trees created by a CMSSW analyzer.

Shell files driving c++ code that run root macros. Sometimes finalize plots using latex.

Personal/group Analysis FRAMEWORK

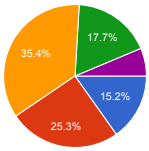
Do you use a personal/group framework to read CMS EDM data and produce private/group root trees?



Yes	**79**	84.9%
No	**14**	15.1%

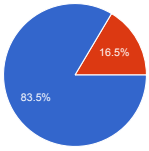
Personal/Group Framework used

What is the origin of the framework?



Personal Framework developed by yourself	**12**	15.2%
Institution Framework used within your institution	**20**	25.3%
A Framework developed and used by a group working on a specific analysis (defined as one or more CADI line, in this case the framework could be used also by more than one institution)	**28**	35.4%
A Framework developed and used within a PAG/POG	**14**	17.7%
Other	**5**	6.3%

Is the framework you use public and available to the CMS collaboration?



Yes	**66**	83.5%
No	**13**	16.5%

If the framework is not a personal framework could you specify which PAG/POG and/or subgroup is responsible for its maintenance?

N/A

-

HIN

CMG

NA

BPH

UCSD/UCSB/FNAL

Unfortunately I don't know...

No specific PAG/POG. Framework mantanied by different institutions

Analysis group for SUS-16-033 (within Inclusive subgroup of SUSY PAG)

Personal Framework of HEPHY Vienna

University of Wisconsin-Madison

FSQ

Muon POG

SMP-VJ

it is personal

I am

cmgtools is used by the CERN group and spans over many PAGs and POGs

Several people are developing Heppy, most of them (I believe) are within the CERN group

JetMET / heppy

mainly maintained by Hbb-folks

SMP-J

It's CMGTools

Korean Collaboration

FSQ PAG

CMGTools group

SMP

Rutgers group

mainly HIG and SUS analyzers

BTV

H -> gamma gamma

Group of authors involved in the CADI lines mentioned before

/

Higgs to Gamma Gamma

Higgs to tautau subgroup

Framework developed and used by some members of the "monojet" analysis team (in the EXO PAG MET+X working group).

DT

HIN, HIN-Dilepton

_

HIN / dilepton

Heavy Ion

SMP EWK

loose association of analyzers

University of Hamburg in B2G

HIN high-pt subgroup

None

At one point I developed the B2G Analysis framework, but we moved away from it. With MINIAOD it became cumbersome and unnecessary. Now we use a simple EDAnalyzer to create TTrees.

HIG

personal framework

It is used by several analyzers in B2G, but B2G itself is not responsible for the framework

Hbb

it's personal framework

UCSB/UCSD/FNAL

it's a personal framework

Exotics (Kevin Sung, Phil Harris, Caterina Vernieri, Nick Wardle)

HIG hbb instance of heppy

B2G/BSM3G

FNAL+UCSB+UCSD groups working on SS, MT2, Stop1L

Notre Dame

The hadronic stop anaylsis group and it is on git SUSYhad2015.

SUSY-photon

University of Wisconsin

It is personal

BPAG

If the Framework is public to CMS could you tell us the repository address?

https://github.com/CERN-PH-CMG/cmgtools-lite

https://github.com/cms-ljmet/Ljmet-Com

https://github.com/cmstas/NtupleMaker

https://github.com/hvanhaev/CommonFSQFramework

https://github.com/cms-analysis/flashgg

https://github.com/CmsHI/cmssw

https://github.com/dntaylor/DevTools

https://github.com/cmkuo/ggAnalysis

https://github.com/TreeMaker/TreeMaker

https://github.com/RazorCMS/SUSYBSMAnalysis-RazorTuplizer

https://github.com/uwcms/FinalStateAnalysis/

https://twiki.cern.ch/twiki/bin/view/CMS/ShearsAnalysisFramework

https://github.com/tvami/cmssw80X-PilotBlade/tree/PilotBladeStudy/

https://github.com/cbernet/cmssw/tree/heppy_8_0_24_patch1

https://github.com/CERN-PH-CMG/cmg-cmssw

github.com/vhbb/cmssw

https://github.com/CERN-PH-CMG/cmgtools-lite/

https://gitlab.cern.ch/shears/shears/tree/master/ntuple_production

https://github.com/cp3-llbb/Framework

git@gitlab.com:Thomassen/RutgersIAF.git

RecoBTag/PerformanceMeasurements/plugins/BtagAnalyzer.cc

https://github.com/pfs/TopLJets2015/tree/master/TopAnalysis

https://gitlab.cern.ch/cms-desy-top/TopAnalysis

CalibMuon/DTCalibration

https://github.com/CMS-HIN-dilepton

https://github.com/CMS-HIN-dilepton/cmssw/tree/Onia_PA_8_0_X/HiAnalysis

https://github.com/qliphy/B2GVV-DATA8X-Vnew

https://github.com/UZHCMS/EXOVVNtuplizerRunII

https://github.com/UHH2/UHH2

https://github.com/CmsHI/cmssw/tree/forest_CMSSW_8_0_22/HeavyIonsAnalysis

https://github.com/cmsb2g/B2GTTbar

https://github.com/NWUHEP/BaconProd

https://github.com/cms-ttH/ttH-TauRoast

https://github.com/vhbb/cmssw/tree/vhbbHeppy80X/VHbbAnalysis

https://gitlab.cern.ch/ncsmith/monoZ

https://github.com/ronchese/NtuTool & https://github.com/ronchese/NtuAnalysis

https://github.com/ksung25/BaconProd

https://github.com/BSM3G

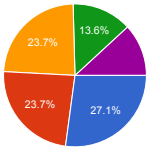
https://github.com/cms-ttH/ttH-13TeVMultiLeptons

https://github.com/CmsHI

https://github.com/susy2015

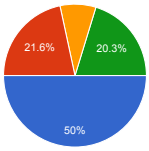
https://github.com/lecriste/Z4430

If you are a developer of the framework could you tell us how many week per year you spend to maintain it? (Implementing new features and/or recommended POG recipes)



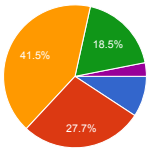
1-2 week/year	**16**	27.1%
3-4 week/year	**14**	23.7%
5-6 week/year	**14**	23.7%
7-8 week/year	**8**	13.6%
more than this	**7**	11.9%

Could you tell us how many times on average you or your group run the framework on the full dataset you use for your analysis (data and MC)? As a reference we suggest to consider the analysis presented at Moriond 2016 and/or ICHEP 2016



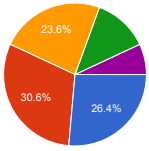
once a month	**37**	50%
twice a month or more	**16**	21.6%
once a week	**6**	8.1%
Other	**15**	20.3%

What is the average time your framework use to process a single event when you run on your data tier (i.e. AOD, miniAOD and/or RECO)?



less than 1 msec (more than 1000 events per seconds are processed)	**6**	9.2%
less than 10 msec (more than 100 events but less than 1000 events per seconds are processed)	**18**	27.7%
less than 100 msec (more than 10 events but less than 100 events per seconds are processed)	**27**	41.5%
less than 1 sec (more than 1events but less then 10 events per seconds are processed)	**12**	18.5%
more than 1 sec (less than 1 events per seconds is processed)	**2**	3.1%

How long it takes to run on the full dataset (data and MC)? Please consider the total number of days including the time needed to run CRAB, collect and/or copy the output files to the final location and any additional post-procession needed.



1-2 day	**19**	26.4%
2 -4 days	**22**	30.6%
4-7 days	**17**	23.6%
8-14 days	**9**	12.5%
more than 2 weeeks	**5**	6.9%

What is the main reason(s) to re-run on the full dataset (data and MC)?

New variables in ntuples/POG recipes

No good reason. Most of the code just copies the miniAOD content. I would say it's mostly a way to make an even smaller tree and have it locally in the T2.

Change of IDs/SFs/Recipes/Rereco

New developments in cmssw, which need to implement in the code. Other new developments in the code.

New features, new recipes from POG that need new inputs not saved before, improvement of the analyses.

Changes in JECs, object IDs, samples (e.g. data ReReco or MC rerun), JSONs, fixing bugs

Add variables or features that were not implemented previously.

Change in JECs, new ID recommendations requiring variables that we have not yet stored, addition of new samples/new data

Changes during Synchronisation

Updating POG recommendations & adding new systematic shifts

New variables added or new recommendations by POGs implemented

New features and improvements

changes in the code yielding, e.g. new variables.

new reco or calibration

Changes in the detector

several groups contribute to the analysis. These groups can decide on a different selection or analysis strategy, there could be a new training for an MVA, etc.

changes in the POG recommendations (JECs, SFs and so on)

update of calibrations / POG objects

changed POG definitions, central SF, GT

new calibrations

Implementation of corrections, new data samples

further corrections, adding variables

To include new objects in the ntuple, change calibrations

upgrade CMSSW version. update JEC or other calibrations. change events or variables to store

Updated data/MC corrections

Store additional information

Update of POG prescriptions / occasional bug

Changes to object ID, jet corrections, MET filter

changes in MC version or JEC

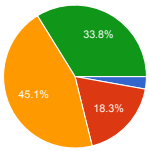
new variables to add to the Ntuples

updates in pre-selection or variable computation (e.g. corrections like JEC)

dedicated energy corrections and smearings

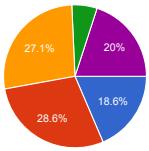
Central re-processing of data/MC
rereco of Data, new MC, changes in POG prescription
New input; or change in recipes that requires full MiniAOD (e.g. all charged particles or recHits in photons)
Synchronisation between two groups to make sure all selections were properly applied.
Updates in the analysis that require additional or modified informations to be stored in the trees.
To get reasonable stat.
Failed jobs, new features, too loose/too tight preselection
incorrect calibrations or updated energy corrections, adding new variables are usually the most common reasons
MC
POG updates
updated conditions / corrections / POG recommendations
Usually every MC coming from CMS is only run once to produce our ntuples.
jet reconstruction details, calibrations change
Modified scale factors (JEC/JER/b-tag/mis-tag)
new calibrations, skim other triggers, add new variables, bug in code
Include/update centrally-provided scale factors and corrections
learn new questions to ask about signal/background separation
Usually updated analysis recipes. We do much pre-processing before the TTrees are made.
Something has changed: CMSSW release, object ID/iso definitions, JEC becomes available, b-tagging scale factors, etc.
updated JECs/ required an object not originally saved in private ntuples
Updates to JEC, event filters, anything needing cmssw implementation
Fast turn-around, output very small on disk
Adding new information or changes caused by updates in the standard prescriptions
New MiniAOD production
New variables, Jet energy corrections, EG regressions
Update recommendations from POGs, fix bugs
cms screws up MiniAOD or is obsessed by the more better for no good reason
Updated calibrations, fixed physics objects, etc.
changing event and/or object selection
Just to clarify: ONE major ReRECO (reported above - 4 months!) followed by lots of >once/week analysis on ntuples (1-2 days)
Well not all of the code was updated. A change in the njets weighting. Some of the rerunning takes like 5 hours and others take 2 days it depends on what needs to be changed.
Fix mistakes and use new implemented algorithms
New data/MC campaigns, missing variables in derived ntuples
Adding new tree branches.
recheck

How many jobs (using CRAB or using local resource) you have to run to complete the full dataset processing (data and MC)?



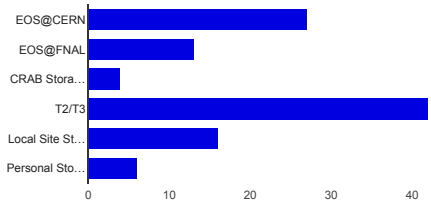
less than 100 jobs	**2**	2.8%
between 100 and 1000 jobs	**13**	18.3%
between 1000 and 10000 jobs	**32**	45.1%
more than 10000 jobs	**24**	33.8%

What is the size of your full processed sample (data and MC)?



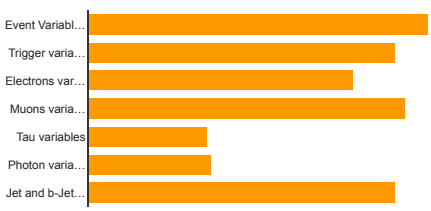
less than 100GB	**13**	18.6%
between 100GB and 1TB	**20**	28.6%
between 1TB and 5TB	**19**	27.1%
between 5TB and 10TB	**4**	5.7%
more than 10TB	**14**	20%

Where do you store the full processed sample (data and MC)?



EOS@CERN	**27**	35.1%
EOS@FNAL	**13**	16.9%
CRAB Storage publishing the result	**4**	5.2%
T2/T3	**42**	54.5%
Local Site Storage (not a T2/T3)	**16**	20.8%
Personal Storage (laptop/desktop)	**6**	7.8%

What are the main quantities you store in your final root trees?

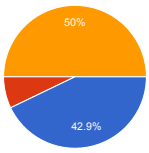


Event Variables (vertex, HT...)	**72**	94.7%
Trigger variables	**65**	85.5%
Electrons variables	**56**	73.7%
Muons variables	**67**	88.2%
Tau variables	**25**	32.9%
Photon variables	**26**	34.2%
Jet and b-Jets variables	**65**	85.5%
MET variables	**50**	65.8%
Generator variables	**60**	78.9%

Other	**10**	13.2%

No Framework used

Would you use a framework which allow you to produce bare root trees with physics quantities already processed if centrally maintained within CMSSW? (providing physics quantities already corrected/calibrated with official POG recipes like, for example, calibrated jets momentum or filtered MET)



Yes	**6**	42.9%
No	**1**	7.1%
Maybe	**7**	50%

Could you describe in few lines how do you produce a plot paper-ready?

Starting from miniAOD objects, we use EDFilters to choose events/objects and an EDAnalyzer to produce ROOT histogram objects. Then a separate PyROOT script combines various datasets to produce paper-ready ROOT canvases.

I run few c++ macro on tof of the flat-Ntuples i produce from the PATtuple

MiniAOD->trees with Heppy, Heppy->plots (some of which may be paper ready) or datacards, datacards -> final plots with Higgs combine/CombineHarvester tool to create postfit distributions if needed. Additional style optimisation with PyROOT scripts if needed.

I write a ROOT macro and sometimes tweak the cosmetics interactively

Standard macros developed by others in my group

I already answered this on page 2. Once the data is in the form of histograms or other TObjects that can be saved to ROOT files, I use ROOT to do the final data analysis if any is required and the final formatting of the plots.

pyroot with a lot of tweaks; input are simple root trees filled from the analysis framework files which are not root-based and contain already all final reconstructed objects & event interpretation

I use a Python script based on the code provided by the CMS Publications Committee, which correctly formats and decorates a canvas according to the accepted style. The objects painted onto the canvas are coded into the script by hand (e.g. histograms, legends, lines, etc).

with root

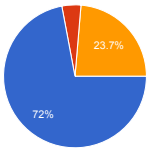
For stack plots, MINI-AOD --> per-event weights --> ROOT skims, then python for plotting

Start with CMSSW to get ntuples and fill histograms and produce cuts interactively. Then plot using a root.

Plots are generally produced with root macros in near-publication form. Occasionally I use latex to create multipanel plots and add some text. Then we go through a multi-month process arguing about minutiae before we shift the text by 1 pixel and call it publication ready.

A tool for official POG recipes cross-check

Do you think that such a tool is useful?



Yes	**67**	72%
No	**4**	4.3%
Maybe	**22**	23.7%

Do you have any comment/suggestion on how to implement this tool?

I would be equally happy to work directly on miniAOD. So, no suggestion...

In addition to the trees, also the code should be made available, so that the analyzers could compare their implementation with this one (not always the twiki pages are updated, but being this a running code it has to be updated and maintained)

I can see the benefit for JEC if that isn't properly applied on miniAOD input, but many of the other corrections as well as overlap removal are highly analysis-dependent (even the way in which b-tag scale factors are applied), and many are not corrections to four-momenta but rather additional event weights. If this is just a tool to check the application of JEC, maybe type-1 MET and lepton momentum corrections for those analyses where the latter two matter, it's not too bad. If it's a tool that does more, say e.g. gets and applies all available corrections and weights from a given POG and hence shows how to apply them, this could be a bit more useful.

It would be nice for such a tool to fit into different analysis frameworks directly. However, it would need to be quite flexible for that... maybe either a simple namespace with boolean functions (returning pass/fail for a single object) or a flexible producer that can output multiple different collections.

output should be as simple as possible and preferably does not need access to CMSSW code and libraries.

For make the initial tree, I have not suggestions. In the Higgs2Tau group we use a synchronization package here: https://github.com/CMS-HTT/2016-sync which has a nice script, compare.py, which runs through 2 TTrees and matches entries by evt# then produces output histograms comparing all identically named variables, i.e. "official recipe" vs. "my recipe".

Similarly to the demo analyzer

not useful because the flat ntuple is very analysis dependent, and because for what is generic, we have mini aods already (which are in the EDM so meta data, etc)

making it enough configurable for all POGs

It sounds extremely useful, also as possibly replacing custom dumpers in the future. Should be implemented with standard CMSSW configurations (cmsRun job), but output to flat (non-CMSSW, non std::vector) trees.

the new miniaod samples are not frequently produced. it is easier/faster work with AOD and a correct global tag

light-weight python script where one can put the branch translations, output either plots or text summary

Functions or classes in Python that don't depend on any specific analysis framework, for example, a function that takes inputs as Python arrays and returns output as Python arrays.

Can be useful for validating a framework. It should be flexible enough to allow checking of each quantity.

This is a great tool for CMS beginners and those doing analyses using "official" objects. For those using "unofficial" objects (I'm thinking here of long lived particles, displaced vertices, heavily ionizing particles, and boosted topologies), this is probably not very useful because analyzers have to invent their own reconstruction and efficiency measurements.

-

We use a non-standard vertex selection (diphoton balancing with tracks) so we'd need to be able to ignore events where our vertex disagreed from MiniAOD default -- including run+event number in the tree would suffice for this

one should think about proper implementation of tools to apply the recipes in the first place

This tool will be very useful provided people spend time in a proper comparison. It may happen that analyzers will have to spend a lot of time in resolving the differences. We have already found that it takes enormous amount of time to synchronize the results of two groups which is a requirement imposed by the Higgs groups.

Do not forget to allow the possibility of a pre-selection

It would be useful in principal but several groups already have their setup and at the moment introducing a new tool would just go unused.

It should be easier to update with the recent recommendation. Also, in some case one need to relax the requirement for e.g. fake photon or electron estimation

this tool is what we've been using so far in the heavy ion group, we start from AOD and produce a collection of TTree's for the group to use, one TTree per physics object collection (i.e. a ttree for tracks, a different ttree for jets, and so on) . Suggestion is to make it as modular as possible, so different sets of variables/trees can be easily turned on and off and run independent of one another. a collection of tools to merge/sort/split these output files could also be useful for different analysis flows.

would be nice if it also works starting from AOD

In my experience, in the past FWLite could be used to do this, but the support for Mac OS X has been spotty enough to make most people move to bare ROOT and TTrees. If we had some quick-and-dirty EDAnalyzer to make TTrees with the right corrections, perhaps most could either use that directly or cut-and-paste the relevant recipes. As of now it seems a bit unwieldy but I didn't attend CMSDAS last week so I cannot make an educated guess how to do that.

THIS WOULD BE EPIC!!!!! The JERC reference table is a good example starting point, I was able to use that table to debug our JEC/JER implementation. A ROOT file (WITH THE CODE THAT MADE IT!) would be even better.

The recipe for cross-checking should be quite simple to implement. I fully understand the problem and feel that it is somewhat ridiculous that we essentially rely on plots for code verification. There should be a better solution, and a cross check is a nice idea, but the recipe will have to be extremely user friendly (or mandated as part of object review) to get people to actually do it

Make it flexible!

I would integrated with miniAOD, i.e. make sure that all POG recipes are integrated and provide solid centrally maintained output. If it's not possible, any format ntuple produced by POG as a reference on using even a single file would work, but integrating it with miniAOD production is 100 times more useful.

It would be extremely useful because it would collect all recipes in a single piece of tagged code, instead of distributing them over several TWikis.

A bare tree can look very different depending on whether it is still structured, or flattened by object, or by event hypothesis. To a large extent, finalized recipes seem to become incorporated into a re-MINIAOD. Existing variables embedded in MINIAOD seem cleaner than recipes in the first place. Also, some recipes must be tuned to an analysis topology's requirements, so this could become difficult.

A good idea, but in practice it might not be widely used if people are not compelled...

make the function part of PAT

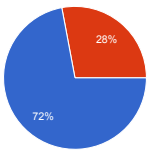
We really need more provenance on all these private trees, but then they'd get too big making them less useful. I do not have a solution, that tool seems interesting, but getting people to do more work other than making plots to use as cross-check is always difficult.

We use Jet tool box for ak8 jets and it works well. There was a problem which we let Jet Tool box people know. The kind of fixed it but there was still an issue we are fixing. Over all I think having already defined POG and PAGS are usefully because you have a starting point for new students as well as even doing a new analysis. Although more complicated analysis will have issues with predefined settings.

This could be a useful resource but only if it is kept extremely up to date (lag of less than 1-2 days between a recipe being announced and it being implemented here). Otherwise many groups will go ahead and implement it separately anyway. Finding a person or group to maintain such a tool at this frequency will be hard, unless somebody uses it for an actual analysis with time pressure from conference deadlines etc.

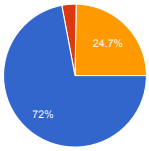
Final questions

Did you follow/Are you following the official POG recipes for ICHEP 2016 (https://twiki.cern.ch/twiki/bin/view/CMS/POGRecipesICHEP2016) for your analysis?



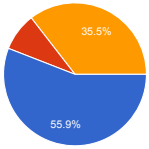
Yes	**67**	72%
No	**26**	28%

Do you think that an official POG recipes for Moriond 2017 twiki page is needed (like the one produced for ICHEP 2016)?



Yes	**67**	72%
No	**3**	3.2%
Maybe	**23**	24.7%

Do you think that a framework centrally maintained within CMSSW which produce bare root trees (with all the official POG recipes) could be useful?



Yes	**52**	55.9%
No	**8**	8.6%
Maybe	**33**	35.5%

What are the main characteristics that you would like to find in such framework?

I don't think that would be very useful. The POGs spend a lot of time developing modules to apply their recommendations on miniAOD. This would be only a repetition of work... and I think that philosophically is the wrong thing to do. We should be asking how to make EVERYONE use miniAOD directly for analysis. Not creating yet another layer.

Simplicity, documentation, be able to customize the physic objects produced

Flexibility, but that would defeat its purpose.

Plain ROOT trees, no dependence on CMSSW libraries, kinematic variables, generator level matching variables, identification variables for objects.

There are various important aspects, and it's difficult to write a one-fits-all solution: - Code should be straightforward and understandable, and not just be a set of configuration files - Needs to be able to submit jobs to all kinds of batch systems that are in use - Should be easily extendible to e.g. allow for an analysis-dependent reconstruction - Needs to have a mechanism to

avoid a large bloat (everyone adding more options) Using Heppy, I think this is in principal a good candidate (even though I suspect some people will not want to use because it's based on Python, so having two frameworks could also be an option). Heppy is used in different ways though, and features that are important to some analyses aren't necessary for others, which leads to a bloat of options and code size (check e.g. the JetAnalyzer). If you don't need all these options, it is much more straightforward to not use the CMS-specific parts of Heppy but just use the generic parts and run a few tasks (say lepton-jet cleaning) manually.

Applies all corrections/calibrations/etc. in a consistent and reliable way and can handle the related systematic uncertainties appropriately

simple to read code so that in case a group finds inconsistencies, it is easy to print out debug statements and to understand where the differences or bugs are coming from. avoid complicated python sequences with lots of interdependencies and implicitly defined variables/names/switches. Those are difficult to track down for debugging purposes. Avoid implementing too many configuration options and don't introduce too much flexibility because this ends up being confusing for readers of the code and will invariably creep towards yet another layer of data format that people run on top of - thus defeating the purpose, which is to reduce the number of data-tiers that occur in a typical workflow.

A good twiki explaining it

Which quantities are saved in the root trees should be configurable via python/json config file

I doubt on this level there exits a good one-fits-all solution

Flat ntuple with no object except std::vector. All POG recipes applied.

I consider heppy to be perfect of course, but the format of the bare root trees should be left to the users. That being said, part of the bare root tree could be centrally defined, the part that is specific to the various physics objects. For example, we could centrally maintain this package: https://github.com/cbernet/cmssw/tree/heppy_8_0_24_patch1/PhysicsTools/Heppy/python /physicsobjects. We could also centrally maintain a root tree in a given format that would be produced on RelVal under the POG recipes, and that the various groups could synchronize to.

Fast, flexible in selecting the variables, able to run on different sites.

runnable on grid, extendable with cmssw modules

easy switching on/off or features, easily extendable e.g. via plugins

easily configurable to get the content you want

literally "bare" root trees, unnecessary to write code like ROOT.gInterpreter.Declare('#some_include.h"') or ROOT.gSystem.Load("libSomething.so")

it should be clear, simple, and flexible

FWLite - based, easy to parallelize on lxBatch

how to get the proper JEC, lepton id, MET, etc...

simplicity, flexibility, it should be easy to configure without touching the code too much

Compactness of the code, small dependencies on other packages, modularity

We would not be able to use this framework because of our unique vertex selection and diphoton treatment, but it could be useful for others

on the fly checking of quantities

The basic kinematic variables, particle ids, events variables, vertex etc. and generator level information for MC data. Also, the trigger information.

A clear definition of the output tree branches and enough flexibility for the user to adapt the tool to a specific output format as desired.

Be able to choose the trigger path and pre-selection

Examples to Implement exactly official recommendation

I'm a bit worried about time-scales. At least for this years Moriond cycle, final corrections/recommendations are coming really late considering the approval timelines. With our current framework we have the choice to a) only implement the recommendations that we actually require b) use preliminary corrections easily (e.g. JECs from files instead of the Database). The other thing that I would like to see in such a framework is easy extendibility, so that it's easy to add additional variables/objects/structures that may be analysis specific.

- It should cover the needs of simple analysis completely. - It should be simple for it to be rerun. - It should be simple to extend it (adding custom event interpretations and store variables). - It should help applying scale factors and their systematic uncertainties (e.g. by pre-calculating and storing eventweights).

Modular, so what ends up in the output trees can be easily controlled by the users. Ability to run on the output you produce to produce a similarly structured output tree with branches taken out (or branches added in).

Well maintained/updated, centrally run

That it just works, and it is properly maintained...

starting from miniAOD plus track error information to allow recomputing vertices, adjustable selection of jet algorithm and cuts, ability to make sequential passes adding new quantities, e.g., initial selection based on leptons, then add jets to those events, then add track block to the previously-selected events.

Applications of JECs and JERs to Jets and MET, up-to-date VIDs for electrons, trigger weights including L1 prescales, generator weights implemented correctly, etc.

Code which abides by best practices for the chosen language(s) and is easily extensible by the end user. Thorough documentation.

This concept might blow up in size, because everyone wants different things... Trigger bits are notoriously hard to store in a small way if you need names attached to them. But I would want at minimum 4 vectors for all physics objects, discriminant values for all common MVAs (b tagging, electron id, etc), trigger flags, met filter flags, isolation components, mini-isolation...it's a long list

It should be written in a modular fashion so that creating modifications requires simply adding a macro, without needing to go into the codebase itself. An example of something similar would be PhysicsModel in Combine. That said, it would also be nice if the codebase were easily comprehensible to the people using the framework, so that it is not a black box to anyone on the collaboration.

The bare root trees have to contain as much information as possible for the analyzers.

Well you will hit the issue of flexibility for different analyses and in the end might just end up with miniAOD equivalent. Also you will probably always have to be remaking a new version of theses trees based on how often recipes change during rushes for conferences

Complex objects (e.g. p4 as a vector class), to not have to manually plug in every single vector component to build geometrical quantities.

Make it part of official miniAOD production. If not possible anything that allows me to check that I get the same numbers (list of objects, their pts etc) is useful.

I think it would quickly become a clone of the MINIAOD data tier

A way to add generalTracks for long lived analyses

Allows for customizable cuts and plotted quantities.

CMS has a terrible track record for making frameworks for analysis. They tend to be cloe to impossible to use.

This framework MUST be easy to checkout AND easy to implement. The trees ideally must be in miniAOD and/or AOD format

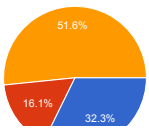
Documentation

Like Jet definitions and objects I think some of it centralized but not all even the naming convention is different between one analysis and another. For new people that is difficult.

It would have to be very well maintained, both for quickly implementing new recipes and also for validation to weed out bugs. It would basically have to be another data tier. Quick turn-around and thorough validation are conflicting goals, though. So I'm skeptical of having a uniform flat ntuple across CMS. Lots of groups have a significant amount invested in their local code bases. As a PhD student on ATLAS a few years ago, I saw them try to implement something similar, flat ntuples that would be produced centrally and used for many analyses. It had multiple problems that eventually made it more of a burden than anything else: variables would be missing, so production would have to be re-run; despite that, there were so many branches that the event size was almost as big or bigger than the EDM data format it replaced; bugs were often introduced and thus propagated to all the dependent analyses; individual analyses had little control on the schedule of production.

Example code

Would you consider to use/switch to such a framework for your analysis?



Yes	**30**	32.3%
No	**15**	16.1%
Maybe	**48**	51.6%

Could you give us few indications on what are the major problems you have to face when running your analysis jobs (using CMSSW, CRAB, your analysis code and/or any other tool used by you for your analysis work)?

CRAB stability followed by large usage of T2s for MC production during campaigns

xrootd reliability

No comments.

CRAB job completion and stageout efficiency needs to improve.

Crab sometimes not properly working. The ntuple files are very large even with HLT skim, takes a lot of time to run on them to produce plots.

Space where to store trees. Last moment recipes with non-standard inputs. Batch system overwhelmed.

There is one major problem: xrootd fails far more frequently than it should. It barely rates one 9 of performance most of the time (i.e. significantly more than 1% of requests fail, and even 1% is fairly high when the full SM MC + 2016 data contains 50K+ files in miniAOD). There is of course a site reliability component in this, but the entire purpose of xrootd (for me as a user) is to get the data to me. I shouldn't have to inspect PhEDEx site lists manually and try to divine the xrootd addresses for different sites in order to get my jobs to connect. (I use HTCondor rather than CRAB because CRAB takes forever to start jobs and return results, and I don't have time to wait around for it. In the time it takes for CRAB jobs to *start* (not to finish!) after submission, I can write, execute, and complete an HTCondor submission for the same jobs, starting from scratch.)

main problems are grid problems. it seems to happen intermittently and maybe more often at busy times like before conference deadlines. causes are typically that some CRAB server or some other critical grid service is overloaded or compromised and CRAB tasks are lost. Then one has to resubmit all the tasks from scratch again. Often crab tasks finish at the 99% level and one job out of 100 repeatedly fails. There are no clear solution to simply resubmit that one failed job. Instead one has to resubmit the entire crab task. Having a way to define a crab task that only runs on parts of the dataset which failed a previous crab task would be a huge improvement. This is possible for data by using json masks, but it would be good to have this possibility in general - also for MC. (because for some signal MC samples, it is necessary to run over ALL files of the MC to get all the parameter space points)

The entire CRAB and cmsweb infrastructure has been somewhat unstable recently. It appears that there is more demand than the resources can handle.

It's all usually pretty smooth, the biggest bottleneck is CRAB when it's having problems/tasks being queued for a long time when everyone is submitting them at the same time. Unavoidable if everyone is trying to do analysis at exactly the same time though!

The largest issue was previously mentioned, it is using the the sv fit algorithm and the second is an mva based MET alog which is not centrally supported.

xrootd access and proof jobs stability. proof-on-demand is not always reliable.

Flaws in the code. The framework works fine

1. GRID reliability: failing jobs and unpredictable availability. 2. Making sure all POG recipes are applied

The pending time is a lot

None

Time in producing the trees

I/O glitchs, hard drive space

failure rates on grid, slow end-to-end iteration time (few ev/s) of codes, custom tools are not interchangeable. Need common CMSSW objects and interfaces everywhere to make modular code (e.g. lightweight replica pat::Electron, pat::Jet)

a lack of instructions, it is hard to find information sometimes, also debug the code wrote by others.

Site problems using crab

storage limitations (e.g. eos allows only < 10000 files), POG recipes require full CMSSW whereas they should be available in FWlite

to get the correct configuration to read the datasets in a correct way

The data are stored as persistent objects of many different classes. This makes it nearly impossible to write a generic code to access to the data stored in objects of all different classes.

data storage, availability of batch resources

CRAB reliability ; boost and python version conflicts ; ROOT Draw slowness

CRAB downtimes due to any number of reasons, running out of personal CERN EOS space when using the CERN batch system to run jobs and write output to EOS

File open errors, making sure official recipes are implemented properly

some jobs have to be resubmit by hand because crab is not able to read the full sample...

random (!) job failures by individual CRAB sites

completing the last few % of the jobs

CPU time because many of the analysis use PF candidates for soft QCD studies in hard events

Better tools for understanding failed CRAB jobs would help. We always get a few failures and it's often hard to tell why. For example, if memory exceeded: did the site hiccup? or are we using a bit too much memory but only sometimes hitting local site limits?

a lot of time is attributed to understand the POG recipes and to distingush them from outdated ones which can even change rapidly w/o code examples (e.g. for Moriond'16 to not propagate JES to MET)

Submitted jobs fail due to many reasons. There is no specific recipe. One has to debug according to a specific situation.

Processing the final trees stored in my institute's T2 is a bit difficult. They take a lot of space. In principle we should be able to access them directly through a dcache system (from a macro used interactively or on a batch system). However accessing in parallel many different sets of trees is too challenging for this system, so there are strong limits on the parallelization. The other solution I found was to copy the trees on /scratch spaces on the UIs, skim them, and keep only the skimmed trees on the UI storage space. I am now considering implementing a (loose) skim directly at the tree production level to optimize this process.

Precisely the fact that they are not bar root trees

CMSSW, CRAB

CRAB downtime

crab could use more dedicated people to take care that it doesnt break often

CRAB sometimes can be slow and one has to wait. Also CERN EOS quota is only 1TB and restrict file numbers to 10K, which make things harder as we are now having more data and MC samples to process.

All of the work is doable. The application of scale factors could be sped up, those are routine-task that take a lot of time.

limited resources (people and hardware)

CRAB works only asymptotically (great management of resubmission, but poor job success rate, need to resubmit jobs multiple time). LSF on lxplus is very crowded. LSF on T2/T3 is best in terms of speed, but has poor job management features, and re-submission is a nightmare. CONDOR could be a good solution but configuration is hard to implement. The optimal solution could be running with LSF within a local T2/T3 on data stored locally (xrootd works fine, but when a failure happens in opening a file, LSF crashes, and resubmission is complicated).

mostly keeping up to date with changes since, e.g., last DAS documentation. Having LPC is real resource for US users who don't get to CERN very often.

Things are much better than the "good ol' days" before MINIAOD, so these are mostly nice-to-have but not critical features. At this point just communicating the recipes to get the right objects is the main bandwidth limiting step, but this is easy to fix.

1 = User error (crap, I ran 5000 jobs with a bug). 2 = job share time (crap, I have to wait 3 days in the queue). 3 = computer down time (crap, I'm at CERN and the LPC nodes just went down so I have to wait 8 hours for people to wake up and restart them)

root is a pain in the butt

Running on CRAB takes long. Some sites are down. There should be a clearer tool to describe which sites are down, for how long, ..?

CRAB error codes not being properly documented.

equivalency of fwlite and cmssw is really the only issue and that is because I inherited a fwlite framework, the modification of which has been easier than conversion to full cmssw

CRAB usage and success is random. Our local resources, despite opportunistic, give faster turnaround while also being reliable. Otherwise, recommendations for custom IDs by analysis

groups have to be re-implemented constantly according to some "recipe", which is just lore. This leads to prolonged debugging periods and syncs. Also, I consider any information on TWikis lost upon being written. Finding information on TWiki without direct links is very hard due to a lot of noise. Plus a lot of pages are completely out of date and misleading. I find this appalling.

Main problem is to implement constantly changing POG recipes. Regarding CRAB - it's unreliable. It happened a number of times that a task can go into some broken state in the crab server and you cannot do anything with it. ASO is another constant source of problems. Overall it's harder to predict the turnaround time at the moment. Nevertheless I think crab3 is overall is a very useful tool with lost of documentation.

Spaghetti-code POG recipes that check out way more CMSSW packages than warranted (especially if they are just changes to python!)

CRAB has been a major bottleneck this year, to the point of analyses missing important deadlines. This week, there were a million jobs queued for 50,000 slots. Tasks are failing frequently due to timeouts or database corruption. CMS needs to consider how to address this problem, whether from the computing side or the user side (e.g. teaching people to write better-performing code, optimize processing, minimize redundancy, etc.).

same as the central production. Tails of processing are the biggest challenge and largest delays.

Large ntuple sizes, given that intermediate root trees are usable by several analyses

@ CERN lxbatch: lack of available resources @CRAB/GRID: weird site errors and small number of jobs that take 10x longer than everything else. CRAB on average pretty reliable @locally (ND): None really, just miscellaneous software bugs

Some percentage of job failures unrecoverable in crab. I had used crab2 for a major production and the book-keeping, job restarting, and data transport was a pain (at the time, not as much xrootd, and transfer between the two T2s led to significant wasted CPU.) So I wrote scripts to handle the file transfer part. Since then a lot of the issues between these particular T2s has been fixed.

Our code seems fine now it takes a little to maintained because we can not just update those parts that are maintained by POGS. Overall it works for now. Starting was a little difficult because I looked at other code and the naming convention was not the same.

CRAB problems can slow things down, also local T2 problems. When new recipes appear for MINIAOD that use variables which have never been used before in standard recipes, that requires re-running and thus wasted time.

Processing time

Consolidation of different analyzers into something coherent.

Do you have any further comment/suggestion on how to improve/simplify the final part of analysis that can be implemented in CMS? (The final part of the analysis is the work needed to produce a public plot starting from a central produced dataset, typically miniAOD)

I think that a better "skeleton maker" for analysis should go a long way. I would like to have a tool that spits out a C++ code that I can immediately start applying selection and making histograms. Now, related to the question above, the official POG/PAG recommendations should be implemented in this "skeleton maker", and not in yet another layer.

N/A

Close collaboration with ROOT team for implementation of new features.

I had a recent discussion about why users don't just run analysis directly on miniAOD (without a flattening step). There are two problems right now preventing that from being an effective solution: 1) As mentioned above, files are frequently not available over xrootd (due to site issues and/or redirector issues). Most T2/T3 sites can't afford to replicate ~100TB of miniAOD samples locally. 2) miniAOD is optimized for compression, not user-friendliness. Interacting with packed classes via FWLite is painful. As an aside, the SUSY PAG had its own ICHEP2016 recommendations page: https://twiki.cern.ch/twiki/bin/view/CMS/SUSRecommendationsICHEP16 Some of that info might be useful to the collaboration as a whole.

The main issue is reliability of the grid. As far as I see, the only reason we perform the final part of the analysis on the CERN batch and NOT on the grid is because the success rate on the CERN batch is roughly 99%, while the success rate on the grid is something like 85-90%. If the grid success rate could be made to approach 99%, then I don't see much reason why anyone would need an intermediate "big" ntuple (few Kb/event). It would be preferable to run EVERYTHING on the grid and just always produce "small" ntuples (0.1Kb/event). However because 10% of jobs fail, it would take at least 2-3 iterations to get most of the jobs back and due to the overhead of submitting jobs, this ends up taking too long. Therefore one opts to go for an intermediate "Big" ntuple. If the grid failure rate can be kept to the 1% level, then I think there can be a huge improvement in efficiency of analysis workflows.

Not at the moment, sorry

Better access to proof resources. proof cluster.

Having more frequent miniaod reprocessing with all POG recipee and latest calibration recommendations applied.

No

common formats for datacards, plotting scripts based on datacards

more information about which systematic uncertainties are necessary to be evaluated to each kind of object, also showing how can be done.

recipes should be available in FWlite and not rely on full CMSSW

I believe that the analysis will be much easier if a centrally produced data set is stored in bare flat trees rather than miniAOD or any EDM formats. Or even further, if it is stored in a more general format such as HDF5, we will have more options for how to analyze and visualize the data.

Something must be done to make it easier to get IDs, corrections, etc. all done correctly

Central review of the coding solutions of POG recipes may improve things since currently those diverge a lot and some are just coded far too overcomplicated which lead to unintentional misuse in the first place (e.g. CMSSW modules involved to apply the JES; recalculate MET). A positive example is the BTagCalibration tool which makes it very clear and easy to test and apply the b-tagging scale factors and even to switch algorithms with close to little effort within CMSSW, FWlite and standalone!

A simple set of common tools will be useful. e.g. a style guide in Root. Often the way the error bars are drawn is a problem with no easy solution.

The idea of a central tool to apply recommended POG recipes is actually an excellent idea and would probably save a lot of time to analyzers in terms of code development. Additionnally, adding informations about Level-1 trigger algorithms that trigger events, and their prescales, in an easily usable format, in the MiniAOD data tier, would be of great help for a number of analysis teams. I know that there are difficulties that are being worked out by the L1 group, and I am sure this will be propagated into the MiniAOD trigger information soon after.

Make data driven methods available ? Or CLs (limit) computation also ?

simplify official recipies to 1 src (C) file, one .h file and one configuration file, create a two example simple analyzers : the one where multiple candidates per event are saved in one root-event and the one with just "by-cancidate-list". Make clear where in the code and how the collections are accesed and what they should correspond to in configuration files.

I always recommend having several steps to go from AOD/RECO to final plot. that way its easier to spot errors/bugs. But such a toolkit would be useful for validation and thats already implemented in the DQM output. so im not super sure what this new framework would be useful for until i see whats in it.

Not for the moment, thanks.

I would be very interested in a tool that allows root-files as a native input for Apache Spark. Hehe. =)

Develop centrally maintained tools for plotting standard features in a common way CMS-wide (data/MC ratio, pulls, ...). Too many times comments on analysis are mostly on aesthetics.

No, this is based on lot on personal needs. If stats committee could make a library of ROOT-style rules for basic things (how to make a pull ratio plot, etc, etc) that would be helpful.

It would be very helpful to have some standard macros/pyROOT scripts for simple but standard plots. Keeping some sort of central online reference for statistics definitions might also be handy.

There should be a central wiki gathering all the information pertinent for the final part of the analysis. For instance, when a problem is spotted during a release validation or in a given dataset, it should be indicated on that wiki.

Its probably best for the experiment as a whole if most individuals have to do this work themselves. it is a very useful thing for students to learn. On the one hand you want to make life easier for people; on the other, if we don't face these types of challenges the ability to write reasonably good code will disappear in a generation or two

Enforce a policy of having custom IDs and recipes in the Framework format: usage should be by including python files and calling a customize function instead of re-implementing a recipe, which tends to be error-prone. We have a great framework, so we should use it!

miniAOD cannot be final format - it's too big and even with skimming you cannot get it small enough for an efficient looping over events. Given that central production can quickly reproduce miniAOD I think we need to make sure that all POG development is implemented there without forcing users to reimplement it.

At some level one must depart from a common starting point. MiniAOD seems to be an excellent improvement for that starting point compared to AOD, but I don't know if there is much room to go further.

miniAOD was optimized for size rather than speed of use. We have our own format for two reasons: 1) speed 2) quality control within our group - we do many analyses, and not all of the

people in our group are equally reliable. To have quality control of what we put out, we depend on a shared fast format and code that analyses that format and is common within our group.

An agreed-upon color scheme could be useful for signal and major backgrounds that appear in stack plots CMS-wide.

At least in HIN, the centralized hiForest got to be very large to incorporate everything needed for each analysis, 4.2 TB for the 2011 PbPb dataset from a 327.8TB RECO input, and that was dropping tracks for space, and we later regretted that choice. Anything to do to narrow down the filesize of centralized ntuples without losing important Physics information is useful.

I like centrals code for something. A lot analysis could use centralized object definitions at least 50% probable more like 80% or 90%. Now yes there is a portion that could not. If the code was written well and maintained than others could go in and change it for there analysis as need be. This would also make it easier to present to POGS/PAGS because you could say specifically what was changed.

Number of daily responses