LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

How to make analysis easy to use, scalable and fast

David Koch

20. 7. 2022

What is this talk about?

- ever increasing luminosity → more data → is the currently established workflow for analyses still feasable?
- Python's ecosystem regarding HEP tools improved vastly during the last years + more and more new physicists don't have knowledge of C++ → being able to complete a full-fleshed analysis using Python only would be desirable

This talk is a broad and short overview about some tools that are available today and ideas about how analyses in the future could look like

Requirement for all future tools, frameworks and resources (in that order!):

easy to use
scalable
fast

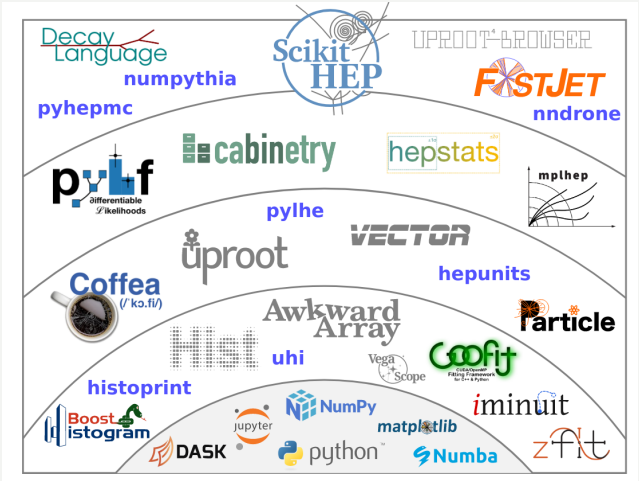
ROOT::RDataFrame

ROOT is the jack of all trades in HEP for basically everything from IO to statistical analysis to plotting

RDataFrame is ROOT's recommended way for working with root files
modern and easy to use declarative API

```
tree = ROOT.RDataFrame("mytree", "path/to/ntuples", daskclient=client)
tree = tree.Define("HT", calc_HT, ["jet_pt"])\
           .Filter("HT > 500")
hist = tree.Histo1D("x")
```

Python ecosystem for analyses



Python ecosystem for analyses

numpy for fast computations with array-like data

```
np.mean(np.random.rand(1000))
```

matplotlib for creating plots

```
plt.hist(distribution, n_bins=20)
plt.title("$p_T$")
```

numba to compile python functions into machine code

```
@numba.njit
def square(x):
  return x**2
```

dask for distributed computing

```
cluster = dask_jobqueue.SLURMCluster(cores=1, queue="ls-schaile")
cluster.scale(4)
client = dask.Client(cluster)
results = client.map(myfunction, data)
```

Python ecosystem for analyses

awkward for numpy-like operations with jagged arrays

```
x = ak.Array([[1., 1.2], [2.], [], [1., 2., 3.]])
x[x**2 > 4.0]  # select entries from x whose square is larger than 4
```

hist and **boost-histogram** as high-level wrappers for `boost::histogram`
zfit for fitting

Python ecosystem for analyses

uproot for IO with ROOT files

```
with uproot.open(filename+":"+treename) as tree:
  for data in tree.iterate(library="ak"):
    # process data ...
```

vector for computations with 4-vectors

```
lepton_p4 = vector.array(dict(pt=lepton_pt, phi=lepton_phi, eta=lepton_eta, M=lepton_m))
```

coffea framework for columnar data analysis in HEP

```
class MyAnalysis(coffea.processor.ProcessorABC):
  def process(self, events):
    selected_electrons = events.electron[events.electron.pt > 25]
    selected_muons = events.muon[events.muon.pt > 25]
    event_filters = ((
      ak.count(selected_electrons.pt, axis=1)
      + ak.count(selected_muons.pt, axis=1)) == 1
    )
    selected_events = events[event_filters]  # ...
```

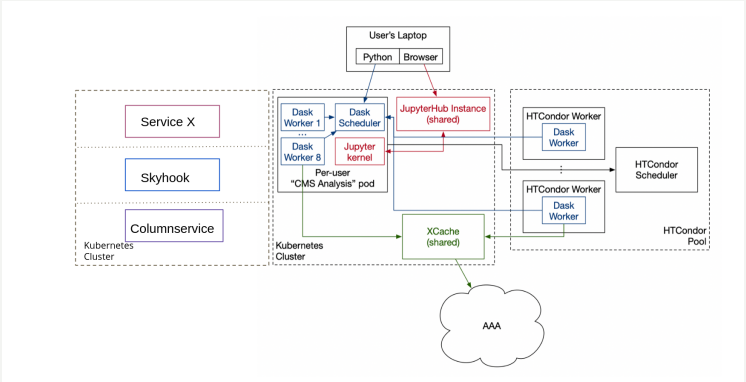
What kind of resources are needed / would be nice to run such code?

Idea of an "Analysis Facility", that provides enough hardware resources to serve analysts who want to run complicated analyses on large datasets

- should be interactive (not like a grid)
- easily scalable
- switching between fully interactive workflow (ie producing plots in < 2 mins on a subset of the data) and execution frameworks (ie Dask or Batch) should be seamless (no more custom batch submission scripts)
- monitoring and performance metrics

Analysis Facilities: Coffea casa

JupyterHub with Dask

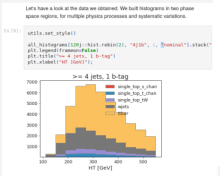


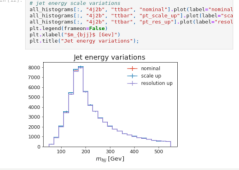
AF@Nebraska (CMS only), AF@Chicago (ATLAS only), AF@Nebraska (open data)

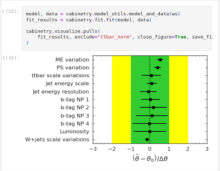
Analysis Grand Challenge

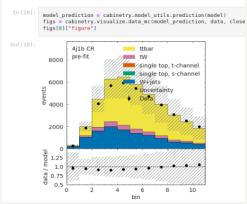
$t\bar{t}$-analysis as a proof of concept of the pythonic tech-stack (→notebook)
focus is on the technical demonstration, not the physics

- meant to be run on ∼ 200 TB of CMS open data
- easily scales up from test runs that can be done on your laptop in a matter of minutes to running on a cluster with Dask
- benchmark of a full run is planned for 2023









The future of analysis?

AFs that provide a JupyterHub are just one approach → not everyone likes to work with jupyter

another vision of future analyses: define the analysis in a domain specific descriptive language and let the rest (ie retrieving the data in an effective way, running in parallel / distributed) be handled automatically → remove the technical aspects of *how* to run the analysis from the physicists and let them focus on the *what* of the analysis
available today: func_adl + ServiceX (partially implemented in the AGC demo analysis)

hot topic of discussion: should we teach new physicists more basics of computer science or should we attempt to make the tools so easy to use that no CS background is needed at all?

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Thank you

for your attention