Columnar data analysis

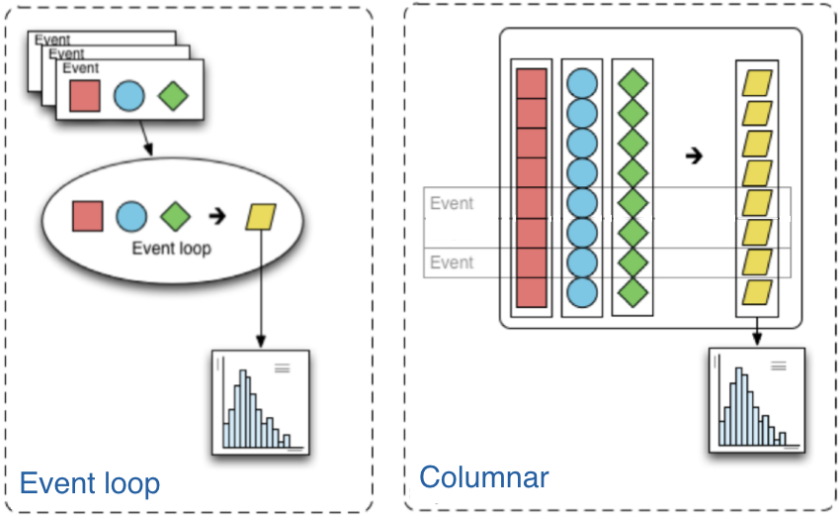
(+ dask)

Nikolai Hartmann

LMU Munich

July 20, 2022, LMU Joint Seminar of Particle Physics Groups

Columnar data analysis



[1] Plot from https://coffeateam.github.io/coffea/concepts.html

Motivation

Operate on columns - *array-at-a-time* instead of *event-at-a-time*

Advantages

- Predefined operations, no for loops!
  → Move slow bookkeeping out of the event loop
  → **Write analysis code in python instead of C++**
- Run on contiguous blocks in memory
  → fast (good for CPU cache, vectorization possible)
- Advances in tools in recent years
  → data science/machine learning
  → also in HEP: uproot, awkward array, coffea

Disadvantages

- Arrays need to be loaded into memory
  → need to process chunk-wise if amount of data too large
- Some operations complex to implement
  (e.g combinatorics, nested selections, variable length lists per event)

But we want

- Objects
  → don't want to manually operate on px, py, pz, ...
- Variable length lists
  → each event has a list of Electrons, Muons, Jets, ...
- Cross references
  → `Electrons.trackParticles.pt` should give me the right thing
  ... even if this is stored in a different column/array

Awkward Array

https://awkward-array.org

- Developed by Jim Pivarski and others from Princeton University
- Supports nested records (`RecordArray`)

  e.g. `Events -> [Electrons -> pt, eta, phi, ..., Jets -> pt, eta, phi ...]`
- Variable length lists (`ListOffsetArray`)
- Cross references via indices (`IndexedArray`)
- Behavior/Dynamic quantities
  e.g. Lorentz vectors - can add vectors, calculate invariant masses, ...
- Everything operates on pure arrays of numbers
  → *structure-of-arrays* instead of *array-of-structs*
- ROOT files via `uproot` , but conversion to/from `ROOT.RDataFrame` also in development

Bring this together with `DAOD_PHYSLITE`

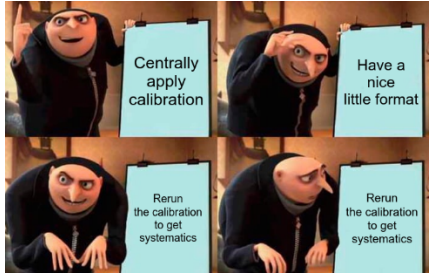
- `DAOD_PHYSLITE`: reduced ATLAS data format with (currently) 10kb per event
  → standard calibrations applied
  → readable (with caveats) without of ATLAS software stack
  → could be used to analyse with python tools and columnar data analysis
  → still in development, many details unclear
- At CMS there is some success with a similar `NanoAOD` format (2kb per event)
- The `coffea` framework provides many functionalities
  → `coffea.nanoevents` for representing such formats as awkward array (including cross references, lazy loading etc)
  → developed prototype schema to support `DAOD_PHYSLITE` with this

Demo

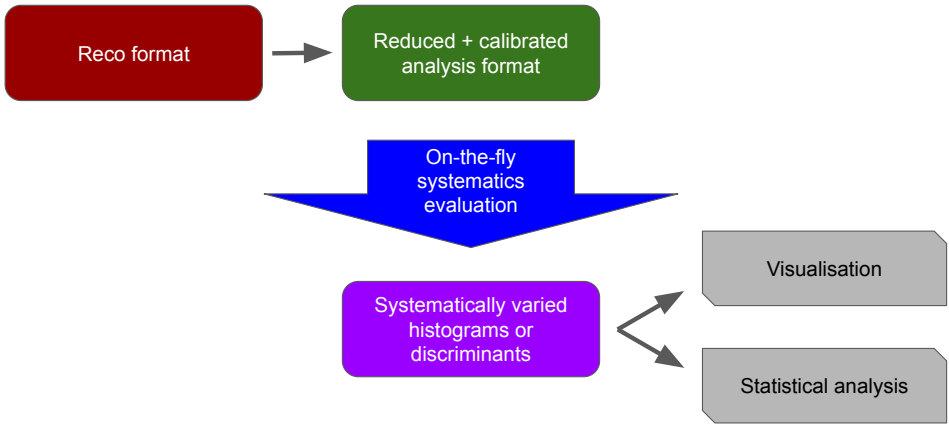
https://github.com/nikoladze/agc-tools-workshop-2021-physlite

Challenge - Systematics

- Vision: evaluate systematic variations on the fly on PHYSLITE
  $\rightarrow$ avoid to store $N_{\text{systematics}}$ copies
- Problem: currently run during calibration (already done in PHYSLITE)
  $\rightarrow$ need to find a way to parametrize based on "nominal" calibration
  $\rightarrow$ ideally not dependent on too many variables
  $\rightarrow$ could also reduce number of needed columns
- At CMS people have apparently managed to do this!



Systematics: The Vision



[1] from Teng Jian Khoo's summary at the Analysis Ecosystems Workshop

Easy to use: ✓
Fast: ✓

next: *Scalable*

Dask

https://www.dask.org

- Python framework for **parallel** computing
- **Low-level** interface via **delayed** and **futures**
  → define custom computation graphs in python
- **High-level**: **distributed** equivalents of **numpy arrays** and **pandas DataFrames**
  → distributed **awkward array** in development
- Live **dashboard** with computations/status/profiler
  → very useful for debugging and optimizing (also looks nice)
- `dask-jobqueue` to spawn dask cluster on top of a **batch system**
  → slurm, HTCondor and more supported
  → single jobs, start immediately with as many workers as you got
- **ROOT RDataFrame** can also use dask as a backend
  → dask as a universal interface to interactive parallelism?

Install via pip or conda

```
# pip
python3 -m pip install dask distributed dask-jobqueue

# conda
conda install dask distributed dask-jobqueue -c conda-forge
```

Easily swap from local to distributed cluster

```
from dask.distributed import Client
# creates a `LocalCluster` with as many workers as cores
client = Client()
```

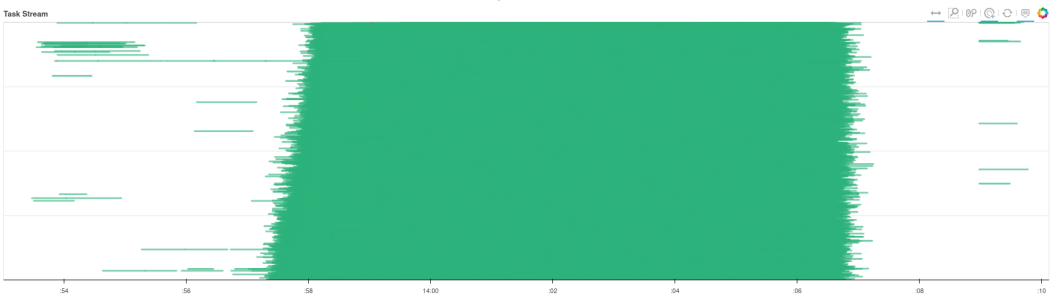
```
from dask.distributed import Client
from dask_jobqueue import SLURMCluster
cluster = SLURMCluster()
client = Client(cluster)
# submit 4 slurm jobs to start workers
cluster.scale(4)
```

→ see ETP wiki for configuration and examples

Google cloud tests

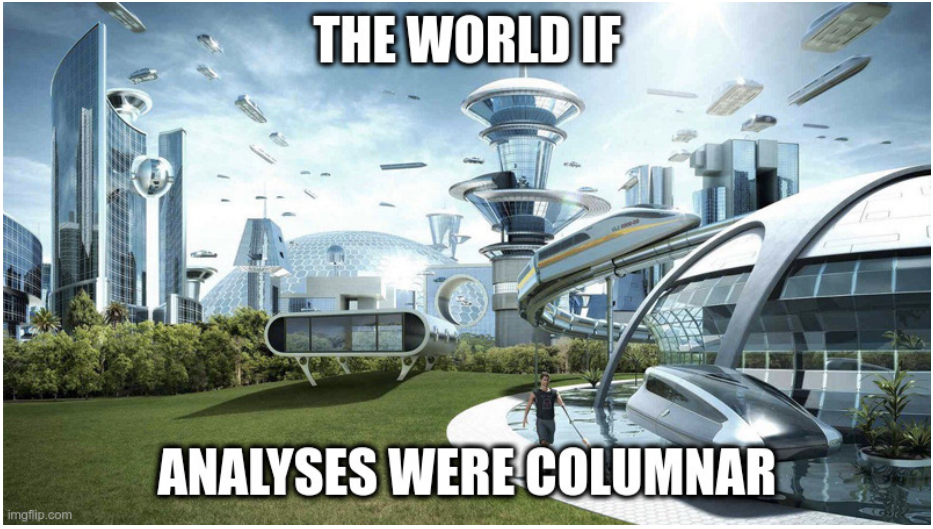
- Tried dask cluster within ATLAS google cloud project
  → advantage of cloud: can scale almost “arbitrary” on-demand
- In principle dask can work with several thousands of cores
  → could process 100TB full Run2 data in 10 minutes
- But: Dasks scheduling model seems to come to limitations
  → seems currently better suited for up to several hundred cores/workers

Task stream plot for 4k workers



Conclusions

- Great benefits from columnar analysis paradigm:
  - Separate bookkeeping from number crunching
  - Analysis code in python
- Already widely used with tabular data
  - candidate-based analysis (e.g. Belle II), global event-based quantities
    $\rightarrow$ `TTree::Draw` is effectively also columnar analysis
  - not so much yet for earlier stages of Analysis
- Tools are available, but handling of systematic uncertainties needs to be sorted out
- Dask might become a universal layer for scaling interactive workflows


THE WORLD IF
ANALYSES WERE COLUMNAR
imgflip.com