bamboo: A high-level HEP analysis library for RDataFrame

27th international conference on Computing in High Energy and Nuclear Physics
Jindrich Lidrych (CP3, UCLouvain) on behalf of the bamboo-hep team

What is bamboo?

A python analysis framework based on ROOT RDataFrame

- A set of tools to efficiently build RDF graphs (JIT-compiled)
- An embedded domain-specific language for producing plots, skim etc.

Design principles

- Analysis code should be as simple and compact as possible
- Be as fast as possible
  → usually one or the other

CMS NanoAOD format + ROOT RDataFrame
→ write physics, not loops

"Hello world" example: dimuon invariant mass

```
from bamboo.analysismodules import NanoAODHistoModule
from bamboo.plots import Plot, EquidistantBinning as EqBin
from bamboo import treefunctions as op

class DimuonPlots(NanoAODHistoModule):
  def definePlots(self, tree, noSel, sample=None, sampleCfg=None):
    plots = []
    if self.isMC(sample):
      noSel = noSel.refine("mcWeight", weight=tree.genWeight)
    muons = op.select(tree.Muon, lambda mu:
                  op.AND(mu.pt > 30., op.abs(mu.eta) < 2.4, mu.mediumId))
    muons = op.sort(muons, lambda mu: -mu.pt)
    twoMuSel = noSel.refine("has2mu", cut=(op.rng_len(muons) > 1 ))
    plots.append(Plot.make1D("dimuM", (muons[0].p4+muons[1].p4).M(),
             twoMuSel, EqBin(100, 20., 120.), title="Invariant mass (GeV)"))
    return plots
```

Main idea: decorate tree

- Decorated version of the input `TTree`: an event looks like a set of containers of physics objects (jets, muons, electrons etc.) and (groups of) per-event quantities (MET, PV, HLT etc.)
- User builds expressions (cut, variables, …) from these python objects
- When done, convert expressions to appropriate (C++) strings, build RDataFrame, run over all samples, and make plots

In the backend: proxies and operations

Operations

- Can be directly converted to C++ strings for JITing
- Simple python objects, immutable – can be modified through a clone, e.g. for systematic variations

Proxies

- Represent objects in the tree, and quantities derived from those
- Behave like the value they represent (list, float, LorentzVector, …)
- Wrap operations
- Automatically generated based on the branches found

- Not specific to the CMS NanoAOD format, nearly any flat tree format may work

bamboo module can look like as

```
class basicPlots(NanoAODHistoModule):
  def addArgs(self, parser):
      ...
  def customizeAnalysisCfg(self, analysisCfg):
      ...
  def prepareTree(self, tree, sample=None,
  sampleCfg=None):
      ...
  def definePlots(self, tree, noSel, sample=None,
  sampleCfg=None):
      ...
  def postprocess(self, taskList, config=None,
  workdir=None, resultsdir=None):
      ...
```

Basic building blocks

Selection object

- Holds cuts and weights
- Start from inclusive selection, unit weight
- Gradually refine selection: add cuts and/or weights
- `RDF::Filter` nodes

Declaring a plot

- Requires only selection object, and plotted quantity
- Fill single or multiple entries (collection)
- `RDF::HistoND` nodes

More advanced functionalities follow same interface

- Selections for data-driven estimations
- Categorized selections

Skims

- `RDF::Snapshot` nodes

Systematic uncertainties

- If an expression is marked as having systematic variations, bamboo will automatically branch the RDF graph
- All systematics are computed on the fly

Event weights

```
from bamboo import treefunctions as op

psFSRSyst = op.systematic(1., name="psFSR",
              up=tree.PSWeight[1], down=tree.PSWeight[2])

selWithSyst = noSel.refine("withFSRSyst", weight=psFSRSyst)
```

Scale factors

- Interface for correctionlib & json file in common CMS format

```
from bamboo.scalefactors import get_correction

muIdSF = get_correction("Muon_SF.json","Muon-ID-SF",
    params={"pt": lambda mu: mu.pt, "eta": lambda mu: mu.eta,
            "year": "2018UL", "WorkingPoint": "Medium"},
    systParam="ValType", systNomName="sf",
    systName="muId", systVariations=("sfup","sfdown"))

twoMuSel = noSel.refine("has2mu", cut=(op.rng_len(muons) > 1 ),
              weight=[muIdSF(muons[0]),muIdSF(muons[1])])
```

Energy scale corrections – CMSJMECalculator

- C++, RDF-friendly standalone package
- Re-apply jet energy correction, smear jet momentum
- Re-calculate missing transverse energy (MET)
- Jets/MET kinematic variations are computed on the fly, automatically propagated to selections & plots

Running an analysis

Running an analysis in bamboo requires:

- An analysis module deriving from a base class (see "Hello world" example)
  → reuse bamboo's facilities for job submissions, sample bookkeeping, etc.
- A configuration `YAML` file with input samples

```
tree: Events
eras:
    2018UL:
      luminosity: 59830.
samples:
    DYJetsToLL_0J_TuneCP5_13TeV-amcatnloFXFX-pythia8:
      era: 2018UL
      db: das:/DYJetsToLL_0J_TuneCP5_13TeV-amcatnloFXFX-pythia8/…/NANOAODSIM
      cross-section: 4757.
      generated-events: genEventSumw
      split: 10
```

Then, just run it: `% bambooRun -m dimuon.py:DimuonPlots dimuon_sample.yml -o myPlots`

Processing modes

Sequential: `% bambooRun ... --distributed sequential`

- Default mode, mostly useful for quick test
- Need to build one RDF graph per sample

Parallel: `% bambooRun ... --distributed parallel`

- Use `RDF::RunGraphs`
- Can use implicit multithreading

Batch: `% bambooRun ... --distributed driver`

- Submit jobs on a cluster (HTCondor, Slurm supported)
- Monitoring loop, combines results for one sample as soon as its jobs are done

Postprocessing

- Write `YAML` config file with list of plots and files, and call plotIt

plotIt:
C++ tool to produce stacked plots using ROOT

bamboo-hep team

- Pieter David – original author
- Oguz Guzel, Khawla Jaffel, Jindrich Lidrych, Sebastien Wertz

Documentation

Features and more

Fairly complete set of features for a typical LHC analysis

- Evaluating MVAs: TMVA, Tensorflow, ONNX Runtime, SOFIE
- Data-driven background estimations
- Making cut flow reports
- Splitting an MC sample into subcomponents
- …