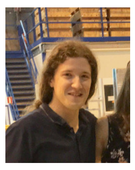


HEP Data Query Challenges

Mason Proffitt, Emma Torró, Gordon Watts
University of Washington, Seattle

Motivation

- Several different code interfaces exist for analyzing HEP data, and new ones are being created
- Each of these analysis description languages (ADLs) needs to be capable of performing common analysis tasks
- The most accessible documentation of an ADL is by working examples inspired by real analyses
- Fair comparison of ADLs requires code samples of each performing the same tasks on the same input

Index repository

- Ideas for common analysis tasks to use as reference were discussed in the HSF Data Analysis Working Group
- An initial list of data query challenges or *functionality benchmarks* was established
- A new GitHub repository was created to document the functionality benchmarks
- Contributors have implemented solutions to the challenges with different ADLs
- The new repository provides an index to these ADLs and links to the implementations of the benchmark tasks

Language catalog

- The functionality benchmarks have been implemented in the following languages so far:

Analysis description language	Programming language	Description
RDataFrame	C++	High-level interface within ROOT for declarative analysis
NAIL (Natural Analysis Implementation Language)	Python	Interface for declarative analysis using RDataFrame as a backend
groot	Go	Pure Go package that provides read/write access to ROOT files
coffea (Columnar Object Framework For Effective Analysis)	Python	Builds on numpy and awkward-array for columnar data analysis

Functionality benchmarks

- The current list of functionality benchmarks is shown in the table below
- The right column explains the generic type of functionality being exemplified
- This will be expanded as new challenges are suggested as GitHub issues to cover missing analysis use cases

Benchmark description	Motivation
Plot the missing E_{T} of all events.	Loop over events and get an event-level variable.
Plot p_{T} of all jets in all events.	Loop over an array in each event.
Plot p_{T} of jets with $\lvert\eta\rvert < 1$.	Loop over an array that is filtered.
Plot the missing E_{T} of events that have at least two jets with $p_{\mathrm{T}} > 40$ GeV.	Loop over an array and aggregate the results to filter at the event level.
Plot the missing E_{T} of events that have an opposite-sign muon pair with an invariant mass between 60 and 120 GeV.	Loop on pairs of objects in one collection and do four-vector algebra.
Plot p_{T} of the trijet system with the mass closest to 172.5 GeV in each event and plot the maximum b-tagging discriminant value among the jets in the triplet.	Loop over combinations of objects in the same collection and extract a property of the combinations other than the key used to sort them.
Plot the sum of p_{T} of jets with $p_{\mathrm{T}} > 30$ GeV that are not within 0.4 in ΔR of any lepton with $p_{\mathrm{T}} > 10$ GeV.	Loop over two different collections.
For events with at least three leptons and a same-flavor opposite-sign lepton pair, find the same-flavor opposite-sign lepton pair with the mass closest to 91.2 GeV and plot the transverse mass of the missing energy and the leading other lepton.	Perform a task for which the formulation in an imperative language is easy but the translation to a functional query language may be less clear or inefficient.

Challenge implementation examples

- An implementation of the first functionality benchmark for each ADL is illustrated below
- All implementations produce a histogram similar to the one at the bottom-right

RDataFrame [1]

```
ROOT::RDataFrame df("Events",
                    "Run2012B_SingleMu.root");
auto h = df.Histo1D({"",
                     ";MET (GeV);N_{Events}",
                     100,
                     0,
                     2000},
                    "MET_pt");
TCanvas c;
h->Draw();
c.SaveAs("met.png");
```

NAIL [2]

```
flow=SampleProcessing("Just MET",
                      "Run2012B_SingleMu.root")
histosPerSelection={"" : ["MET_pt"]}
flow.printRDFCpp([],
                 debug=False,
                 outname="tmp.C",
                 selections=histosPerSelection)
os.system("g++ -fPIC -Wall -O3 tmp.C $(root-
    config --libs --cflags) -o tmp")
os.system("./tmp 4")
```

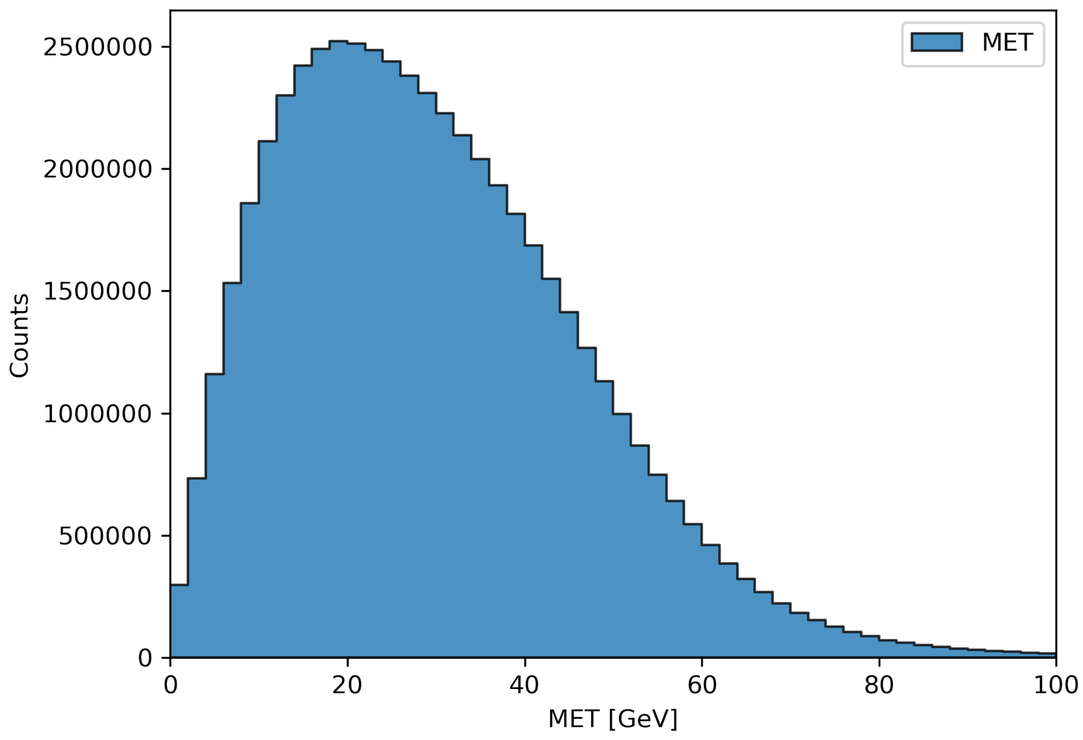
groot [3]

```
f, err := groot.Open("Run2012B_SingleMu.root")
o, err := f.Get("Events")
tree := o.(rtree.Tree)
var hmet = hbook.NewH1D(100, 0, 2000)
sc, err := rtree.NewTreeScannerVars(tree, rtree
    .ScanVar{Name: "MET_pt"})
for sc.Next() {
        var met float32
        err := sc.Scan(&met)
        hmet.Fill(float64(met), 1)
}
p := hplot.New()
p.X.Label.Text = "MET [GeV]"
p.Y.Label.Text = "Nevts"
p.Add(hplot.NewH1D(hmet))
err = p.Save(10*vg.Centimeter, -1, "met.png")
```

coffea [4]

```
class METProcessor(processor.ProcessorABC):
    def __init__(self):
        dataset_axis = hist.Cat("dataset", "")
        MET_axis = hist.Bin("MET", "MET [GeV]", 50, 0, 100)
        self._accumulator = processor.dict_accumulator({'MET': hist.
            Hist("Counts", dataset_axis, MET_axis)
        })
    @property
    def accumulator(self):
        return self._accumulator
    def process(self, df):
        output = self.accumulator.identity()
        output['MET'].fill(dataset=df['dataset'], MET=df['MET_pt'].
            flatten())
        return output
    def postprocess(self, accumulator):
        return accumulator
fileset = {'MET': ["Run2012B_SingleMu.root"]}
output = processor.run_uproot_job(fileset, treename='Events',
    processor_instance=METProcessor(), executor=processor.
    futures_executor, executor_args={'workers':4}, chunksize = 500000)
hist.plot1d(output['MET'], overlay='dataset', fill_opts={'edgecolor':
    (0,0,0,0.3), 'alpha': 0.8})
```

Output



Summary

- A list of functionality benchmarks designed to demonstrate how to perform common analysis tasks in an analysis description language has been created
- Several contributors have written implementations of these benchmark tasks in different analysis languages
- A repository documenting the functionality benchmarks, as well as indexing analysis languages and their implementations of the benchmarks has been created at `https://github.com/iris-hep/adl-benchmarks-index`

References

[1] Stefan Wunsch. `https://github.com/stwunsch/opendata-benchmarks`.

[2] Andrea Rizzi. `https://github.com/arizzi/nail/tree/master/benchmarks`.

[3] Sebastien Binet. `https://github.com/go-hep/examples/tree/master/groot/bench-opendata`.

[4] Mat Adamec. `https://github.com/mat-adamec/coffea-benchmarks`.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Cooperative Agreement OAC-1836650. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.