

FemtoCode: querying HEP data

Jim Pivarski

Princeton University – DIANA

April 17, 2017

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

For us, this would mean being able to do final analysis directly on collaboration-shared Analysis Object Datasets (AODs), without managing private skims.

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

For us, this would mean being able to do final analysis directly on collaboration-shared Analysis Object Datasets (AODs), without managing private skims.

However, these systems don't deal (well) with rich objects, like arbitrary-length lists of jets containing lists of tracks.

(I last talked about this on December 12.)

Query systems

In some industries, it is commonplace to query petabytes of data in real time, usually with SQL. (Ibis, Impala, Kudu, Drill, etc.)

For us, this would mean being able to do final analysis directly on collaboration-shared Analysis Object Datasets (AODs), without managing private skims.

However, these systems don't deal (well) with rich objects, like arbitrary-length lists of jets containing lists of tracks.

Femtocode

I'm developing a query system whose performance permits real-time analysis, but is capable of complex manipulations, such as filtering tracks, picking pairs to compute invariant masses, etc.

Language/compiler

- ▶ As familiar to the user as possible (objects, nested loops).
- ▶ But constrained to allow restructuring for fast execution (map/filter/reduce instead of for loops, total-functional).
- ▶ Extra-strength type system to eliminate runtime errors.

Execution engine

- ▶ Operate on contiguous columns of data (like “TLeaf”), not objects. Restructuring becomes changes in arrays of integers.
- ▶ No memory allocation at runtime, vectorizable loops.
- ▶ JIT-compiled. CPU target for now, but GPU is possible.

Distributed server

- ▶ Vending machine: queries go in, histograms (etc.) come out.
- ▶ Referential transparency eliminates the need for “sessions.”

```

pending = session.source("ZZ_13TeV_pythia8")
    .define(mumass = "0.105658")      # chain of operations on source
    .toPython(mass = "")

muons.map(mu1 => muons.map({mu2 =>   # doubly-nested loop over muons
    plx = mu1.pt * cos(mu1.phi);
    ply = mu1.pt * sin(mu1.phi);     # shares scope with other steps
    plz = mu1.pt * sinh(mu1.eta);    # in the chain (see "mumass")
    E1 = sqrt(plx**2 + ply**2 + plz**2 + mumass**2);

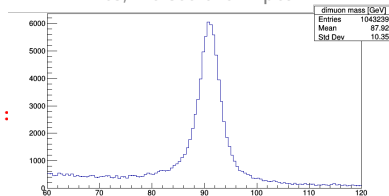
    p2x = mu2.pt * cos(mu2.phi);
    p2y = mu2.pt * sin(mu2.phi);
    p2z = mu2.pt * sinh(mu2.eta);
    E2 = sqrt(p2x**2 + p2y**2 + p2z**2 + mumass**2);

    px = plx + p2x; py = ply + p2y;
    pz = plz + p2z; E = E1 + E2;

    # "if" is required to avoid sqrt(-x)
    if E**2 - px**2 - py**2 - pz**2 >= 0:
        sqrt(E**2 - px**2 - py**2 - pz**2)
    else:
        None      # output type is nullable
}))
""").submit()      # asynchronous submission to
final = pending.await()      # watch result accumulate

```

Yes, we see the Z peak.



- ▶ Femtocode always appears in quotes (like SQL). It is a big-data aggregation step in the midst of a traditional analysis.
- ▶ A query is a “workflow” from source to aggregation, compiled and submitted as one unit.

e.g. `source("dataset").define(X).define(Y).histogrammar(Z)`

- ▶ Most Femtocode expressions are tiny (hence “femto”), scattered throughout a Histogrammar aggregation:

```
session.source("dataset")
  .define(goodmuons = ""..."")
  .filter("goodmuons.size >= 2")
  .define(dimuon = ""..."")
  .bundle(                                # plot the dimuons we've defined
    mass = bin(120, 0, 12, "dimuon.mass"),
    pt = bin(100, 0, 100, "dimuon.pt"),
    eta = bin(100, -5, 5, "dimuon.eta"),
    phi = bin(314, 0, 2*pi, "dimuon.phi + pi"),
    # also plot one muon per entry
    muons = explode("goodmuons", "mu", bundle(
      pt = bin(100, 0, 100, "mu.pt"),
      eta = bin(100, -5, 5, "mu.eta"),
      phi = bin(314, -pi, pi, "mu.phi"))))
```