Jim Pivarski's update

Jim Pivarski

Princeton University – DIANA

September 25, 2017

Overview

Strategy

Everything that I'm working on is intended to increase the "separation of concerns" between physics/statistical analysis and computing issues, so that data analysts can focus on data.

Overview

Strategy

Everything that I'm working on is intended to increase the "separation of concerns" between physics/statistical analysis and computing issues, so that data analysts can focus on data.

Tactics

The idea of adding abstraction layers is not new, but physicists don't often use them because they're unfamiliar, slow, or poorly advertised. Therefore, as a means to the above, I'm. . .

Overview

Strategy

Everything that I'm working on is intended to increase the "separation of concerns" between physics/statistical analysis and computing issues, so that data analysts can focus on data.

Tactics

The idea of adding abstraction layers is not new, but physicists don't often use them because they're unfamiliar, slow, or poorly advertised. Therefore, as a means to the above, I'm. . .

1. developing projects in stages so that small improvements can be rolled out before radically new interfaces;

Overview

Strategy

Everything that I'm working on is intended to increase the "separation of concerns" between physics/statistical analysis and computing issues, so that data analysts can focus on data.

Tactics

The idea of adding abstraction layers is not new, but physicists don't often use them because they're unfamiliar, slow, or poorly advertised. Therefore, as a means to the above, I'm. . .

1. developing projects in stages so that small improvements can be rolled out before radically new interfaces;
2. ensuring high performance so that speed can be the first selling point to attract physicists to use these products;

Overview

Strategy

Everything that I'm working on is intended to increase the "separation of concerns" between physics/statistical analysis and computing issues, so that data analysts can focus on data.

Tactics

The idea of adding abstraction layers is not new, but physicists don't often use them because they're unfamiliar, slow, or poorly advertised. Therefore, as a means to the above, I'm...

1. developing projects in stages so that small improvements can be rolled out before radically new interfaces;
2. ensuring high performance so that speed can be the first selling point to attract physicists to use these products;
3. lagging behind in advertising my products, largely because the software itself is not ready yet. But soon!

Overview as a timeline

2016 projects

- **Data into Spark** for CMS Big Data Project
  - became roo4j/spark-root, completed by Viktor
  - 2017 summer student: Pratyush Das contributed
- **Histogrammar:** functional histograms
  - primary way to do HEP-style plotting in Spark (CMS BDProj)
  - I intend to repurpose it for querying HEP data (below)

2017 projects

- **Computing on columnar data** to rapidly query HEP data
  - collaboration with Tanu Malik on database indexing of hierarchical, columnar data
- **Distributed query server** for HEP analysis
  - collaboration on LDRD with Oli and Igor
  - 2017 summer student: Thanat Jatuphattharachat investigated
- **Contributions to ROOT and NanoAOD:** Numpy interface, performance tests, reading NanoAOD outside of C++, uproot

2018 and beyond

- **Femtocode:** query language for HEP data (deferred)

Computing on columnar data

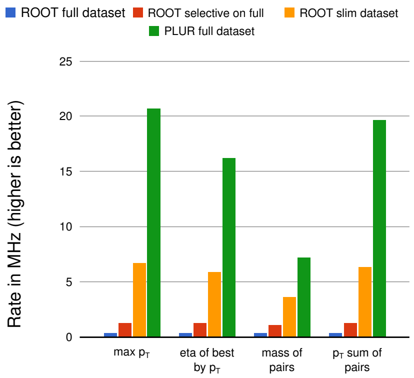
We've been splitting hierarchical data into columns *for storage* for many years now. The new idea is to do calculations directly on columnar arrays, without reconstructing events and particles first.

Computing on columnar data

We've been splitting hierarchical data into columns *for storage* for many years now. The new idea is to do calculations directly on columnar arrays, without reconstructing events and particles first.

```
# objects in Python code
def dimuon(event):
  n = len(event.muons)
  for i in range(n):
    for j in range(i+1, n):
      m1 = event.muons[i]
      m2 = event.muons[j]
      mass = sqrt(
        2*m1.pt*m2.pt*(
        cosh(m1.eta - m2.eta) -
        cos(m1.phi - m2.phi)))
      fill_histogram(mass)

# translated to array references
plur.compile.run(arrays, dimuon)
```

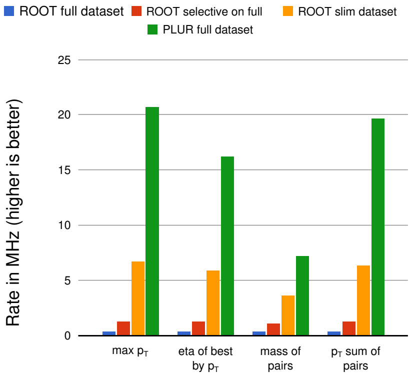


Computing on columnar data

We've been splitting hierarchical data into columns *for storage* for many years now. The new idea is to do calculations directly on columnar arrays, without reconstructing events and particles first.

```
# objects in Python code
def dimuon(event):
  n = len(event.muons)
  for i in range(n):
    for j in range(i+1, n):
      m1 = event.muons[i]
      m2 = event.muons[j]
      mass = sqrt(
        2*m1.pt*m2.pt*(
        cosh(m1.eta - m2.eta) -
        cos(m1.phi - m2.phi)))
      fill_histogram(mass)

# translated to array references
plur.compile.run(arrays, dimuon)
```



Submitted to IEEE Big Data; subject of my StrangeLoop talk.

Sharing data among versions

With an open API for *managing* data with columnar granularity (see PLUR project), new versions of data structures can be produced without modifying or copying the original data.

Sharing data among versions

With an open API for *managing* data with columnar granularity (see PLUR project), new versions of data structures can be produced without modifying or copying the original data.

For instance, a flat list of events

```
    [Event, Event, Event,          Event, Event, Event, Event]
```

can be turned into a nested list of lumi blocks that partition events

```
[Lumi([Event, Event, Event]), Lumi([Event, Event, Event, Event])]
```

without touching the Event columns! (Only adding Lumi columns, including an offset array to *conceptually* restructure the events.)

Sharing data among versions

With an open API for *managing* data with columnar granularity (see PLUR project), new versions of data structures can be produced without modifying or copying the original data.

For instance, a flat list of events

```
    [Event, Event, Event,         Event, Event, Event, Event]
```

can be turned into a nested list of lumi blocks that partition events

```
[Lumi([Event, Event, Event]), Lumi([Event, Event, Event, Event])]
```

without touching the Event columns! (Only adding Lumi columns, including an offset array to *conceptually* restructure the events.)

Users can correct jet energies, add reconstructed candidates, link gen and reco, skim particles, events, etc., all without modifying or copying— so that dataset versions can share content like git.

Database indexing

Collaboration with Tanu Malik, database expert in the CS department at DePaul.

In particular, she's interested in the idea of applying database-style indexes to *hierarchical* (non-tabular) data.

Database indexing

Collaboration with Tanu Malik, database expert in the CS department at DePaul.

In particular, she's interested in the idea of applying database-style indexes to *hierarchical* (non-tabular) data.

Some ideas: Approximate indexing with zonemaps and bitmaps, sorting events to concentrate high-p_T in fewer disk pages. . .

Database indexing

Collaboration with Tanu Malik,
database expert in the CS department
at DePaul.

In particular, she's interested in the idea
of applying database-style indexes to
hierarchical (non-tabular) data.

Some ideas: Approximate indexing with zonemaps and bitmaps,
sorting events to concentrate high-p_T in fewer disk pages. . .

Tanu found prior art in the literature:
Toni Mattis, Johannes Henning, Patrick Rein, Robert Hirschfeld, and Malte Appeltauer.
2015. *Columnar objects: improving the performance of analytical applications.*
DOI: http://dx.doi.org/10.1145/2814228.2814230

(But it doesn't do code transformation or database-style indexing.)

Database indexing

Collaboration with Tanu Malik, database expert in the CS department at DePaul.

In particular, she's interested in the idea of applying database-style indexes to *hierarchical* (non-tabular) data.

Some ideas: Approximate indexing with zonemaps and bitmaps, sorting events to concentrate high-p_T in fewer disk pages. . .

Tanu found prior art in the literature:
Toni Mattis, Johannes Henning, Patrick Rein, Robert Hirschfeld, and Malte Appeltauer. 2015. *Columnar objects: improving the performance of analytical applications.* DOI: http://dx.doi.org/10.1145/2814228.2814230
(But it doesn't do code transformation or database-style indexing.)

We're also writing a community whitepaper on these topics.

Goals for columnar computing

Short term (∼few weeks)

Finish the "PLUR" tools for physicists to run Python functions on their local, skimmed data faster than conventional C++ methods (`SetBranchAddress/GetEntry`).

- Read ROOT with Brian's BulkIO through my Numpy interface.
- New columns, and someday database indexes, in SSD cache.
- Transform Python code with PLUR, compile with Numba.
- Use BulkIO/PLUR-friendly data tiers like CMS's NanoAOD.

Goals for columnar computing

Short term (∼few weeks)

Finish the "PLUR" tools for physicists to run Python functions on their local, skimmed data faster than conventional C++ methods (`SetBranchAddress/GetEntry`).

- Read ROOT with Brian's BulkIO through my Numpy interface.
- New columns, and someday database indexes, in SSD cache.
- Transform Python code with PLUR, compile with Numba.
- Use BulkIO/PLUR-friendly data tiers like CMS's NanoAOD.

Long term (∼1 year)

Demonstrator of a HEP query service: physicists submit Python code to a server, it runs on many nodes in parallel and returns small objects (histograms). Should be $\mathcal{O}(1\text{ second})$ to be a reasonable competitor to local skims.

- working with Oli and Igor, incorporating ideas from Thanat's summer study, still hoping to work with off-the-shelf pieces. . .

Contributions to ROOT and CMS

ROOT: Numpy interface on top of BulkIO

- Gets data into Scientific Python libraries (Pandas, machine learning) more rapidly, not only PLUR.
- Full integration with PyROOT (done).
- High-level connectors to common Python libraries (planned).

Contributions to ROOT and CMS

ROOT: Numpy interface on top of BulkIO

- Gets data into Scientific Python libraries (Pandas, machine learning) more rapidly, not only PLUR.
- Full integration with PyROOT (done).
- High-level connectors to common Python libraries (planned).

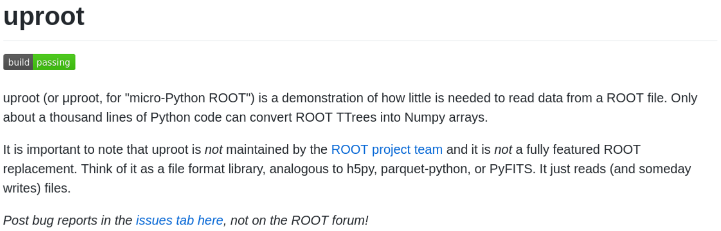
CMS: Testing CMS NanoAOD

- To ensure it's BulkIO-capable and PLUR-capable.
- Performance-testing NanoAOD with BulkIO, compression (LZ4), and other experimental features like Brian's offset removal. Useful feedback to both NanoAOD group and ROOT I/O.
- It would be great if there was a similar movement in ATLAS; the DxAODs are all `std::vectors` (destructively serialized, yet with the same semantic information as a `TLeaf[N]`).

Alternative ROOT reader

The BulkIO-Numpy API is simple enough (no streamers or classes) that it can be implemented in pure Python, without ROOT as a dependency.

https://github.com/scikit-hep/uproot



uproot

build passing

uproot (or μproot, for "micro-Python ROOT") is a demonstration of how little is needed to read data from a ROOT file. Only about a thousand lines of Python code can convert ROOT TTrees into Numpy arrays.

It is important to note that uproot is *not* maintained by the ROOT project team and it is *not* a fully featured ROOT replacement. Think of it as a file format library, analogous to h5py, parquet-python, or PyFITS. It just reads (and someday writes) files.

Post bug reports in the issues tab here, not on the ROOT forum!

1. View whole ROOT file as a memory-mapped array for rapid seeking.
2. Find bytes corresponding to basket data and cast as Numpy arrays.
3. Similar performance as BulkIO (same technique), but easier to install.
4. Small scope: just reading simple leaves, mirroring ROOT-Numpy API.

Conclusions

I'm working on many different projects, with many different people, but these projects are all connected: we're developing a high-level analysis environment with a performance *gain*, not a cost.

Conclusions

I'm working on many different projects, with many different people, but these projects are all connected: we're developing a high-level analysis environment with a performance *gain*, not a cost.

Staged approach: first developing tools for analysts to use on their existing skims, then providing the same interface in a distributed service, then vectorization/type-checking gains from Femtocode.

Conclusions

I'm working on many different projects, with many different people, but these projects are all connected: we're developing a high-level analysis environment with a performance *gain*, not a cost.

Staged approach: first developing tools for analysts to use on their existing skims, then providing the same interface in a distributed service, then vectorization/type-checking gains from Femtocode.

Combination of uproot (for BulkIO-Numpy style interface), PLUR, and Histogrammar should be physicist-ready in weeks, maybe a month. Then I'll respond to feedback and iterate.