Data Analysis R&D

Jim Pivarski

Princeton University – DIANA-HEP

February 5, 2018
Eventual goal

**Query-based analysis:** let physicists do their analysis by querying a central dataset instead of downloading and managing private skims. *Remove an expensive middleman!*

Existence proof

1. I’ve worked with such systems at several large companies as a statistical consultant, regularly querying terabytes in seconds.

2. Just to prepare this talk, I ran a query on Google BigQuery (2 TB in 30 sec). Analysis-as-a-service is common in industry: this one is publicly accessible.
<table>
<thead>
<tr>
<th>Row</th>
<th>timestamp</th>
<th>country_code</th>
<th>file_name</th>
<th>file_version</th>
<th>type</th>
<th>url</th>
<th>details_distro_name</th>
<th>details_distro_version</th>
<th>det</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2017-11-24 23:26:53.000 UTC</td>
<td>null</td>
<td>uproot-2.2.tar.gz</td>
<td>2.0.2</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.0.2.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>2</td>
<td>2017-11-24 23:02:03.000 UTC</td>
<td>null</td>
<td>uproot-2.2.tar.gz</td>
<td>2.0.2</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.0.2.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>3</td>
<td>2017-11-24 23:02:03.000 UTC</td>
<td>null</td>
<td>uproot-2.2.tar.gz</td>
<td>2.0.2</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.0.2.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>4</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>5</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>6</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>7</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>8</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>9</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>10</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>11</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>12</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>13</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>14</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>15</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>16</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
<tr>
<td>17</td>
<td>2017-11-30 14:25:20.000 UTC</td>
<td>null</td>
<td>uproot-2.1.tar.gz</td>
<td>2.1.5</td>
<td>sbt</td>
<td>packages/m4/0.6.10/0/c05a765e57577a7206e7717552a026a064316873a04a8b513096/uproot-2.1.5.tar.gz</td>
<td>Raspbian GNU/Linux</td>
<td>9</td>
<td>Lin</td>
</tr>
</tbody>
</table>
These systems are not suited for HEP data, but they could be

<table>
<thead>
<tr>
<th>Source data format</th>
<th>Google/Big Data</th>
<th>HEP</th>
<th>what I’m developing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parquet, ORC,</td>
<td>ROOT</td>
<td><strong>uproot</strong>: array-oriented</td>
</tr>
<tr>
<td></td>
<td>Avro, BSON, . . .</td>
<td></td>
<td>ROOT reader</td>
</tr>
<tr>
<td>Query language</td>
<td>SQL</td>
<td>Python or C++</td>
<td><strong>oamap</strong>: columnar objects</td>
</tr>
<tr>
<td>Distributed storage</td>
<td>GFS/HDFS</td>
<td><em>similar</em> (Ceph?)</td>
<td><strong>starting now</strong></td>
</tr>
<tr>
<td>Distributed processing</td>
<td>Dremel/Drill</td>
<td><em>similar</em> (Dask? Zookeeper?)</td>
<td><strong>future</strong></td>
</tr>
<tr>
<td>User interface</td>
<td>web dashboard,</td>
<td>TDataFrame,</td>
<td><strong>Histogrammar</strong>:</td>
</tr>
<tr>
<td></td>
<td>Google Sheets,</td>
<td>PyROOT, Jupyter,</td>
<td>functional histogramming,</td>
</tr>
<tr>
<td></td>
<td>REST queries</td>
<td>SWAN, Spark</td>
<td><strong>Femtocode</strong> (<em>future</em>)...</td>
</tr>
</tbody>
</table>
This talk: pieces that can be used on their own, right now

**uproot:** pure-Python ROOT reader that directly copies columnar ROOT data into Numpy arrays.

**oamap:** object-array mapping (analogy to ORM) translating processes defined on virtual objects into operations on columnar arrays.
Numpy is the reason an ML ecosystem developed in Python

Standardized way of wrapping low-level (fast) arrays in high-level (convenient) Python.

Most scientific Python libraries are compiled code with Python interfaces, and Numpy is the standard way to move or share data between them.

uproot provides this kind of access to ROOT.
Load one attribute for all events at a time

```python
>>> import uproot
>>> t = uproot.open("tests/samples/Zmumu.root")["events"]
>>> t.keys()
['Type', 'Run', 'Event', 'E1', 'px1', 'py1', 'pz1', 'pt1', 'et1', 'phi1', 'Q1', 'E2', 'px2', 'py2', 'pz2', 'pt2', 'et2', 'phi2', 'Q2', 'M']

>>> t["M"].array()
array([ 82.46269156,  83.62620401,  83.30846467, ...,  95.96547966,
       96.49594381,  96.65672765])

>>> t.arrays(["px1", "py1", "pz1"])
{'py1': array([ 17.433243, -16.5703623, -16.5703623, ...,  1.1994057,
               ...,  1.1994057])}

>>> t.arrays()  # all of them!
...
Doing meaningful calculations with Numpy arrays

```python
>>> import uproot, numpy
>>> t = uproot.open("tests/samples/Zmumu.root")["events"]
>>> px, py, pz = t.arrays(["px1", "py1", "pz1"], outputtype=tuple)
>>> # compute pt for all events in the first pass
>>> pt = numpy.sqrt(px**2 + py**2)
>>> # compute eta for all events
>>> eta = numpy.arctanh(pz / numpy.sqrt(px**2 + py**2 + pz**2))
>>> # compute phi for all events
>>> phi = numpy.arctan2(py, px)
>>> print(pt, eta, phi)
```
```
[ 44.7322 38.8311 38.8311 ..., 32.3997 32.3997 32.5076 ],
[-1.21769 -1.05139 -1.05139 ..., -1.57044 -1.57044 -1.57078 ],
[ 2.74126 -0.44087 -0.44087 ..., 0.03702 0.03702 0.036964]
```

The loop over events is in (fast) compiled code, not (slow) Python for loops. When we iterate over big datasets, we iterate in batches:

```python
uproot.iterate("files*.root", "events", ["px1"], entrystep=1000)
```
Connector to an external package: Pandas

$ pip install pandas --user

```python
>>> df = t.pandas.df()
# all the same arguments as t.arrays()
>>> df

   E1      E2  Event       M       Q1  Q2  Run     \n0  82.201866  60.621875 10507008  82.462692  1  -1  148031
1  62.344929  82.201866 10507008  83.626204  -1  1  148031
2  62.344929  81.582778 10507008  83.308465  -1  1  148031
3  60.621875  81.582778 10507008  82.149373  -1  1  148031
...
2302  1.199406  -26.398400  -74.532431  -153.847604 GT
2303  1.201350  -26.398400  -74.808372  -153.847604 GG
```

[2304 rows x 20 columns]

Pandas is a Swiss army knife for in-memory, tabular data analysis. Linked to plotting tools, exploratory data analysis, and machine learning. The popular Python for Data Analysis book is basically a Pandas tutorial.
Data with non-uniform width (not scalar numbers)

```python
>>> t = uproot.open("tests/samples/mc10events.root")['Events']
>>> a = t.array("Muon.pt")  # such as std::vector<numbers>
    # variable length for each event

jaggedarray([[ 28.07074928],
              [],
              [ 5.52336693  5.4780116  4.13222885],
              ...
              [],
              [ 6.85138178],
              []])
```
Data with non-uniform width (not scalar numbers)

```python
>>> t = uproot.open("tests/samples/mc10events.root")["Events"]
>>> a = t.array("Muon.pt")  # such as std::vector<numbers>
>>> a
# variable length for each event
jaggedarray([[ 28.07074928],
[,],
[ 5.52336693 5.4780116 4.13222885],
..., [,],
[ 6.85138178],
[]])
```

```python
>>> for event in a:
...    for muon in event:
...        # conceptually, an array of different-length arrays
...```
Data with non-uniform width (not scalar numbers)

```python
t = uproot.open("tests/samples/mc10events.root")['Events']
a = t.array("Muon.pt") # such as std::vector<numbers>
>>> a
# variable length for each event

jaggedarray([[ 28.07074928], []],
            [ 5.52336693  5.4780116  4.13222885], ...
            [],
            [ 6.85138178], []])

>>> a.contents # but efficiently stored as a contiguous block
array([ 28.07074928,  5.52336693,  5.47801161,  4.13222885, ...
       5.06344414,  6.85138178], dtype=float32)

>>> a.stops # with event boundaries in a separate array
array([ 1,  1,  4,  7,  7,  8, 13, 13, 14, 14])
```
>>> import uproot
>>> f = uproot.open("mixed-data.root")
>>> f.allclasses()  # list object names and classes (recursively)
{'gaussian;1': <class 'uproot.rootio.TH1F'>, 'events;1': <class 'uproot.rootio.TTree'>}

>>> f["gaussian"].show()

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2411</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-inf, -3)</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>-3, -2.4)</td>
<td>755</td>
<td></td>
</tr>
<tr>
<td>-2.4, -1.8)</td>
<td>285</td>
<td></td>
</tr>
<tr>
<td>-1.8, -1.2)</td>
<td>1580</td>
<td></td>
</tr>
<tr>
<td>-1.2, -0.6)</td>
<td>2296</td>
<td></td>
</tr>
<tr>
<td>-0.6, 0)</td>
<td>2286</td>
<td></td>
</tr>
<tr>
<td>0, 0.6)</td>
<td>1570</td>
<td></td>
</tr>
<tr>
<td>0.6, 1.2)</td>
<td>795</td>
<td></td>
</tr>
<tr>
<td>1.2, 1.8)</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>1.8, 2.4)</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>2.4, 3)</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Other features

- Explicit caching (user-provided \texttt{dict} or \texttt{dict}-like object)
- Explicit parallel-processing (user-provided \texttt{ThreadPoolExecutor})
- Optional non-blocking calls (useful when parallel-processing)
- Lazy arrays (load on demand, aware of TBasket structure)
- Numba integration: operations on Numpy arrays, JaggedArrays and TH1 can be compiled (more later)
- Functional chains (like TDataFrame)
pip install uproot --user

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

http://uproot.readthedocs.io
uproot download statistics (pip logs are in Google BigQuery)

- **Announcement:**
  - Version 2.0 announcement at HSF meeting
  - Total so far: 347 downloads

- Count of country_code:
  - US, FR, HU, GB, DE, CH, CA, IT, FI, NO, IN, KR, AU, JP, NL, TR, OM, RO

- Count of platform:
  - Arch Linux, CentOS Linux 7, Fedora 24, LEDE 17.014, macOS 10.11.6, macOS 10.13, macOS 10.8.5, Raspbian GNU/Linux 8, Scientific 6.9, Scientific Linux 7.4, Scientific CERN SLC 6.9, Ubuntu 16.04

- Map showing downloads by country.
JaggedArrays are only one data structure: list of lists of numbers.

- Implementation as an array of numbers with array of boundaries is much more efficient than thousands of tiny arrays randomly scattered in memory.
- Sublists are “views” created on demand. (Not created at all in compiled code!)
JaggedArrays are only one data structure: list of lists of numbers.

- Implementation as an array of numbers with array of boundaries is much more efficient than thousands of tiny arrays randomly scattered in memory.
- Sublists are “views” created on demand. (Not created at all in compiled code!)

**Generalize to a complete typesystem:**

- **primitives:** booleans, numbers, characters— anything fixed-width;
  - **lists:** arbitrary length sequences of any single type;
  - **unions:** logical-or type (e.g. a Particle is an Electron or a Photon);
  - **records:** logical-and type: “struct” object that contains named, typed fields;
  - **tuples:** like a record, but typed fields are indexed, not named;
  - **pointers:** redirect to another collection to make trees, graphs, enumerations;
  - **extensions:** runtime interpretations of the above (e.g. list of chars as a “string”).
```python
>>> import oamap.source.parquet
>>> stars = oamap.source.parquet.open("planets*.parquet")
>>> stars

[<Star at index 0>, <Star at index 1>, <Star at index 2>,
 <Star at index 3>, <Star at index 4>, ...]

>>> stars[0].ra, stars[0].dec

(293.12738, 42.320103)

>>> stars[258].planets

[<Planet at index 324>, <Planet at index 325>, <Planet at index 326>,
 <Planet at index 327>, <Planet at index 328>]

>>> [x.name for x in stars[258].planets]

["HD 40307 b", "HD 40307 c", "HD 40307 d", "HD 40307 f", "HD 40307 g"]
```
Explore, scan with compiled code, then explore some more...

```python
>>> import numba  # compiles array-based Python code
>>> import oamap.compiler  # loads object-array compiler extensions
>>> @numba.njit  # decorator for compiling a function
>>> def orbital_period_ratio(stars):
...     out = []
...     for star in stars:
...         best_ratio = None
...         for one in star.planets:
...             for two in star.planets:
...                 ratio = one.orbital_period.val / two.orbital_period.val
...                 if best_ratio is None or ratio > best_ratio:
...                     best_ratio = ratio
...                 if best_ratio is not None and best_ratio > 200:
...                     out.append(star)
...     return out
>>> extremes = orbital_period_ratio(stars)
>>> extremes
[<Star at index 284>, <Star at index 466>, <Star at index 469>, ...
```

17 / 20
oamap is an infrastructure component

- TBranch data efficiently lifted from ROOT into arrays, efficiently scanned with oamap/numba-compiled functions.
- Columns of data are never turned into objects.
  - No need for schema evolution (no container classes).
  - ROOT-style selective and contiguous branch reading at all stages of the calculation: disk → memory and memory → CPU.
  - Alternate between object-oriented operations and vectorized (or GPU) operations.
  - Manipulate structure of dataset without copying data.
  - Different dataset versions can share the majority of their columns.
- These techniques are common for SQL in databases, but new to full programming environments like Python objects.
- There was nothing special about Python; this could be done for C++ as well.
pip install oamap --user

https://github.com/diana-hep/oamap

(The demo examples use a Parquet file of NASA Exoplanets because I want it to be accessible to developers outside of HEP, to attract outside help.)
Final slide: reminder of how these fit into a larger project

<table>
<thead>
<tr>
<th>Source data format</th>
<th>Google/Big Data</th>
<th>HEP</th>
<th>what I’m developing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parquet, ORC, Avro, BSON, ...</td>
<td>ROOT</td>
<td>uproot: array-oriented ROOT reader</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query language</th>
<th>SQL</th>
<th>Python or C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>oomap: columnar objects</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distributed storage</th>
<th>GFS/HDFS</th>
<th>similar (Ceph?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>starting now</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Distributed processing</th>
<th>Dremel/Drill</th>
<th>similar (Dask? Zookeeper?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>future</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User interface</th>
<th>web dashboard, Google Sheets, REST queries</th>
<th>TDataFrame, PyROOT, Jupyter, SWAN, Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogrammar: functional histogramming, Femtocode (future) ...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>