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.

Google BigQuery

Google BigQuery

COMPOSE QUERY

Query History

Job History

Filter by ID or label

HEPQuery

No datasets found in this project.

Please create a dataset or select a new project from the menu above.

Public Datasets

- bigquery-public-data:hacker_news
- bigquery-public-data:noaa_gsod
- bigquery-public-data:samples
- bigquery-public-data:usa_names
- gdelt-bq:hathitrustbooks
- gdelt-bq:internetarchivebooks
- lookerdata:cdc
- nyc-tlc:green
- nyc-tlc:yellow

New Query

Query Editor | UDF Editor

```sql
SELECT timestamp, country_code, file.filename, file.version, file.type, url, details.distro.name, details.distro.version, details.system.name, details.system.release, details.cpu
FROM TABLE_DATE_RANGE(
  [the-psf:pypi.downloads],
  TIMESTAMP("2017-01-20"),
  CURRENT_TIMESTAMP()
)
WHERE file.project="uproot" and details.installer.name == "pip" and details.distro.name == "Raspbian GNU/Linux"
```

Ctrl + Enter: run query, Tab or Ctrl + Space: autocomplete.

RUN QUERY | Save Query | Save View | Format Query | Show Options

Query complete (30.2s elapsed, 1.82 TB processed)

Results | Details

Download as CSV | Download as JSON | Save as Table | Save to Google Sheets

Row	timestamp	country_code	file_filename	file_version	file_type	url	details_distro_name	details_distro_version	det
1	2017-11-24 23:26:53.000 UTC	null	uproot-2.0.2.tar.gz	2.0.2	sdist	/packages/f4/06/91b0cc7d9b5ef9cb071e72b8ce77195baa22a6c99c431680a144ab858026/uproot-2.0.2.tar.gz	Raspbian GNU/Linux	9	Linu
2	2017-11-24 23:02:03.000 UTC	null	uproot-2.0.2.tar.gz	2.0.2	sdist	/packages/f4/06/91b0cc7d9b5ef9cb071e72b8ce77195baa22a6c99c431680a144ab858026/uproot-2.0.2.tar.gz	Raspbian GNU/Linux	9	Linu
3	2017-12-11 21:51:53.000 UTC	null	uproot-2.5.9.tar.gz	2.5.9	sdist	/packages/7d/b0/ecf6134c205e2678ca28fa749330357631322163849c017b2fd635822291/uproot-2.5.9.tar.gz	Raspbian GNU/Linux	8	Linu
4	2017-11-30 14:15:25.000 UTC	null	uproot-2.1.5.tar.gz	2.1.5	sdist	/packages/96/1c/f246d9beae81fbad3fc1d0bd675a0516b4286c23b9afefe92f3099bf5f6d/uproot-2.1.5.tar.gz	Raspbian GNU/Linux	8	Linu
5	2017-11-30 13:35:34.000 UTC	null	uproot-2.3.3.tar.gz	2.3.3	sdist	/packages/5d/8b/dc6486334876c81c23034e60dd836fe9b2bf7eb4e5b39df1a83e1e8aaa23/uproot-2.3.3.tar.gz	Raspbian GNU/Linux	8	Linu
6	2018-01-03 23:50:16.000 UTC	null	uproot-2.5.15.tar.gz	2.5.15	sdist	/packages/7a/1f/c4fcb3b32809f45c03c3d7adfb1051176b6ad35c104040fc9fa38f47b44c/uproot-2.5.15.tar.gz	Raspbian GNU/Linux	8	Linu
7	2018-01-03 16:47:34.000 UTC	null	uproot-2.5.14.tar.gz	2.5.14	sdist	/packages/e8/65/858fd40c433c9a1034fa3062897ff3aa19a58c2d80d596fdbbf6df573c2f/uproot-2.5.14.tar.gz	Raspbian GNU/Linux	8	Linu
8	2017-11-04 02:11:29.000 UTC	null	uproot-1.0.1.tar.gz	1.0.1	sdist	/packages/d3/50/4af1773d449a454ee31e20278ec114bceba1bc473b636a11685d40b6552f/uproot-1.0.1.tar.gz	Raspbian GNU/Linux	8	Linu
9	2017-11-04 02:11:54.000 UTC	null	uproot-1.4.2.tar.gz	1.4.2	sdist	/packages/e1/9b/6ae6cc054af7e61fecc7cfb45b7a18c0f3c06dcfd1a596ca2cb844f84a93/uproot-1.4.2.tar.gz	Raspbian GNU/Linux	8	Linu
10	2017-11-04 02:12:01.000 UTC	null	uproot-1.6.0.tar.gz	1.6.0	sdist	/packages/98/0c/7a7f934b1e7a9a302e8ef16edc030ba17e5759141ed9e86aaefbb0464734/uproot-1.6.0.tar.gz	Raspbian GNU/Linux	8	Linu
11	2017-11-04 02:11:55.000 UTC	null	uproot-1.5.3.tar.gz	1.5.3	sdist	/packages/60/14/bfa043c970b66b6f73fe0765ffdeec5964338c2a29301cc2a1c7740be209/uproot-1.5.3.tar.gz	Raspbian GNU/Linux	8	Linu
12	2017-11-04 02:11:55.000 UTC	null	uproot-1.5.0.tar.gz	1.5.0	sdist	/packages/b7/12/3fd299b2fd9c2076d491933ac7a1177ade4a0d715fbc56beb8e14c91cf66/uproot-1.5.0.tar.gz	Raspbian GNU/Linux	8	Linu
13	2017-11-04 02:11:27.000 UTC	null	uproot-1.0.0.tar.gz	1.0.0	sdist	/packages/5c/0a/7ca1f587d460cb8fe70306a94308822b2cce5bf176cd4f0879246950c73f/uproot-1.0.0.tar.gz	Raspbian GNU/Linux	8	Linu
14	2017-11-04 02:11:29.000 UTC	null	uproot-1.2.0.tar.gz	1.2.0	sdist	/packages/c0/e1/c249b359c0dfce204968f6fefff84813199b77b66a43b36727c4d2ed60f4/uproot-1.2.0.tar.gz	Raspbian GNU/Linux	8	Linu
15	2017-11-04 02:11:39.000 UTC	null	uproot-1.3.0.tar.gz	1.3.0	sdist	/packages/7e/6d/f070aff4821b78a8bf7d133922ad93540c82a01924473e1d77149d6fb9be/uproot-1.3.0.tar.gz	Raspbian GNU/Linux	8	Linu
16	2017-11-04 02:12:05.000 UTC	null	uproot-1.6.1.tar.gz	1.6.1	sdist	/packages/d7/39/8eadf6a705cd6b6b778fc2b067d920b0776bb1b3b11e3359f55d786640cb/uproot-1.6.1.tar.gz	Raspbian GNU/Linux	8	Linu
17	2017-11-04 02:11:40.000 UTC	null	uproot-1.3.1.tar.gz	1.3.1	sdist	/packages/1b/b2/c59d1916ca1aa631f2264114cbc4ac16a226ac705821a2ab48e6bb16a352/uproot-1.3.1.tar.gz	Raspbian GNU/Linux	8	Linu

Table | JSON

First < Prev Rows 1 - 17 of 37 Next > Last

These systems are not suited for HEP data, but they could be

	Google/Big Data	HEP	what I'm developing
Source data format	Parquet, ORC, Avro, BSON, . . .	ROOT	**uproot:** array-oriented ROOT reader
Query language	SQL	Python or C++	**oamap:** columnar objects
Distributed storage	GFS/HDFS	*similar* (Ceph?)	*starting now*
Distributed processing	Dremel/Drill	*similar* (Dask? Zookeeper?)	*future*
User interface	web dashboard, Google Sheets, REST queries	TDataFrame, PyROOT, Jupyter, SWAN, Spark	**Histogrammar:** functional histogramming, **Femtocode** *(future)*. . .

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

```
>>> import uproot
>>> t = uproot.open("tests/samples/Zmumu.root")["events"]
>>> t.keys()

['Type', 'Run', 'Event', 'E1', 'px1', 'py1', 'pz1', 'pt1',
 'eta1', 'phi1', 'Q1', 'E2', 'px2', 'py2', 'pz2', 'pt2',
 'eta2', '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,
 ...

>>> t.arrays()     # all of them!

 ...
```

Doing meaningful calculations with Numpy arrays

```
>>> 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:

```
uproot.iterate("files*.root", "events", ["px1"], entrystep=1000)
```

Connector to an external package: Pandas

```
$ pip install pandas --user
```

```
>>> df = t.pandas.df()    # all the same arguments as t.arrays()
>>> df
                E1          E2      Event           M  Q1  Q2     Run   \
0        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)

```
>>> 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],
             []])
```

Data with non-uniform width (not scalar numbers)

```
>>> 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],
             []])

>>> for event in a:
...     for muon in event:
...         # conceptually, an array of different-length arrays
...
```

Data with non-uniform width (not scalar numbers)

```
>>> 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])
```

Not just TTrees; reads TStreamers and interprets several types

```
>>> 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()

                   0                                                  2411
                   +---------------------------------------------------+
[-inf, -3)     0   |                                                   |
[-3, -2.4)     68  |*                                                  |
[-2.4, -1.8)   285 |******                                             |
[-1.8, -1.2)   755 |****************                                   |
[-1.2, -0.6)   1580|**********************************                 |
[-0.6, 0)      2296|************************************************   |
[0, 0.6)       2286|***********************************************    |
[0.6, 1.2)     1570|*********************************                  |
[1.2, 1.8)     795 |****************                                   |
[1.8, 2.4)     289 |******                                             |
[2.4, 3)       76  |**                                                 |
[3, inf]       0   |                                                   |
                   +---------------------------------------------------+
```

Other features

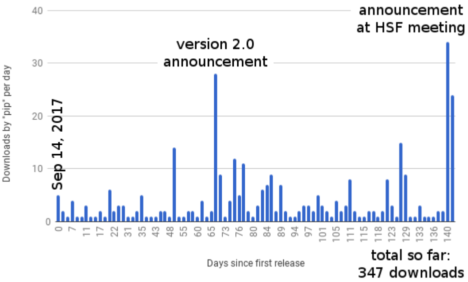
- Explicit caching (user-provided `dict` or `dict`-like object)
- Explicit parallel-processing (user-provided `ThreadPoolExecutor`)
- Optional non-blocking calls (useful when parallel-processing)
- Lazy arrays (load on demand, aware of TBasket structure)
- Numba integration: operations on Numpy arrays,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

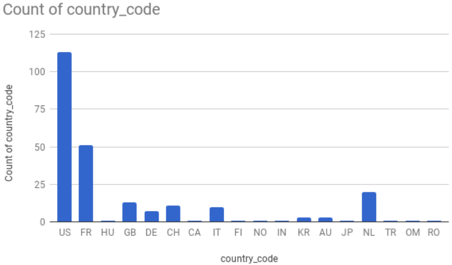
```
pip install uproot --user
```

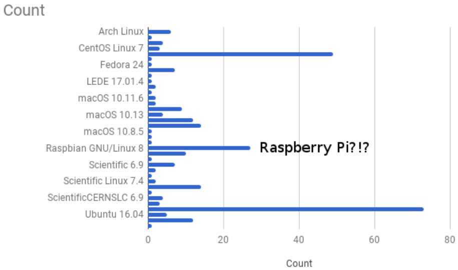
https://github.com/scikit-hep/uproot

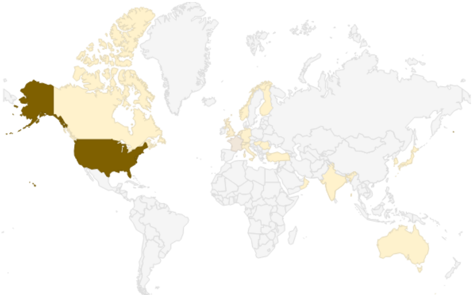
http://uproot.readthedocs.io

uproot download statistics (pip logs are in Google BigQuery)









Object-Array Mapping: generalizing JaggedArrays

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!)

Object-Array Mapping: generalizing JaggedArrays

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").

NASA Exoplanets example on website

```
>>> 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. . .

```
>>> 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>, ...
```

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 $\rightarrow$ memory and memory $\rightarrow$ 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

	Google/Big Data	HEP	what I'm developing
Source data format	Parquet, ORC, Avro, BSON, ...	ROOT	**uproot:** array-oriented ROOT reader
Query language	SQL	Python or C++	**oamap:** columnar objects
Distributed storage	GFS/HDFS	*similar* (Ceph?)	*starting now*
Distributed processing	Dremel/Drill	*similar* (Dask? Zookeeper?)	*future*
User interface	web dashboard, Google Sheets, REST queries	TDataFrame, PyROOT, Jupyter, SWAN, Spark	**Histogrammar:** functional histogramming, **Femtocode** *(future)*...