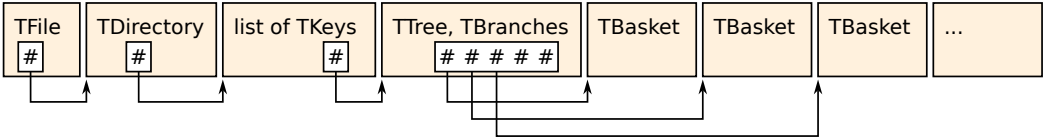
Tiled-Uproot: a use of Awkward Arrays in Tiled and a possible Tiled Adapter

Jim Pivarski

Princeton University – IRIS-HEP

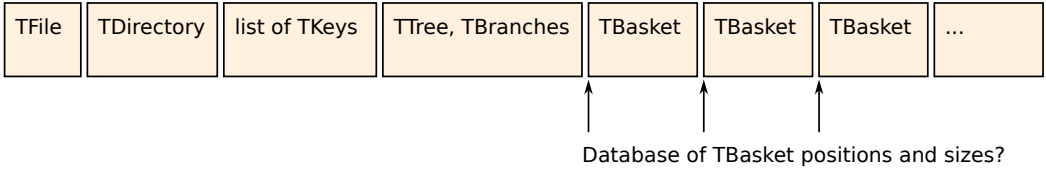
October 25, 2023

Finding data in a ROOT file involves several round-trips

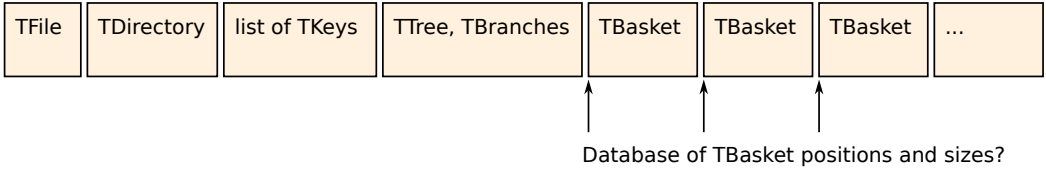


1. Request initial bytes of ROOT file.
2. With the response, find the root TDirectory and request its bytes.
3. With that response, find its list of TKeys and request its bytes.
4. With that response, find the desired TTree and request its bytes.
5. With that response, find all desired TBaskets and request all of their bytes.

Finding data in a ROOT file involves several round-trips



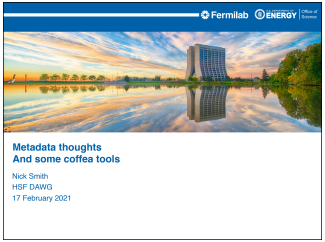
Finding data in a ROOT file involves several round-trips



Why would you want to do that?

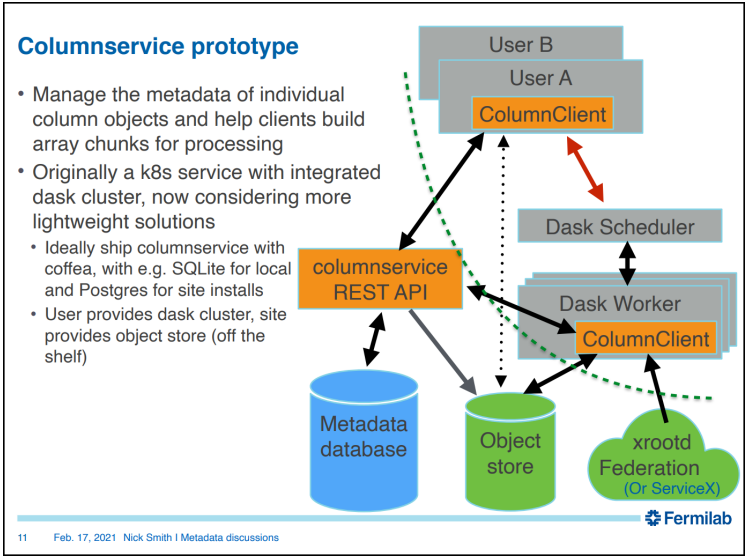
- Replace the latency of 4 round trips with the latency of a (nearby) database.
- Specialized TTree/RNTuple access for large sets of files.
- Leave a set of ROOT files in place, so they don't need to be converted and can be accessed in traditional ways as well.
- Can be made to look the same as a database of user-contributed Awkward Arrays.

Nick Smith's columnservice prototype



https://github.com/CoffeaTeam/columnservice

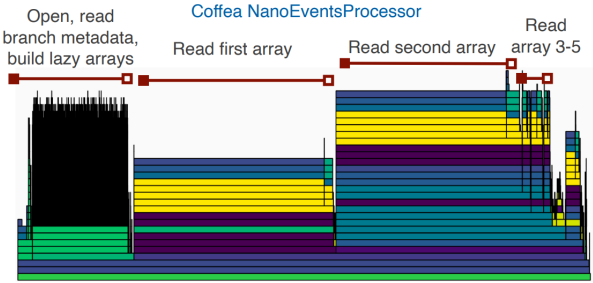
(last activity: Mar 2021)



Nick Smith's columnservice prototype

Columnservice case study: avoiding ingestion

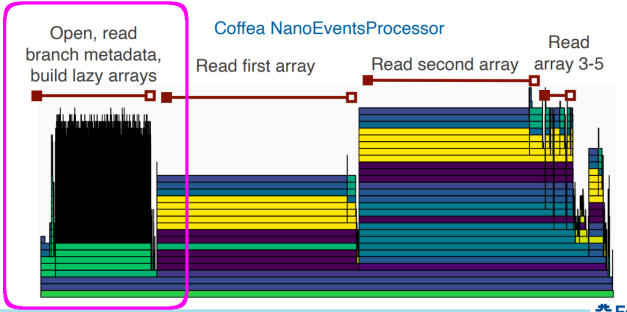
- All inputs eventually come from ROOT files
  - True for the foreseeable future
- Reading and interpreting files with uproot is expensive
  - Even just opening and getting branch names can be significant
  - File byte-range caches take time to kick in, bad for small work packages



Nick Smith's columnservice prototype

Columnservice case study: avoiding ingestion

- All inputs eventually come from ROOT files
  - True for the foreseeable future
- Reading and interpreting files with uproot is expensive
  - Even just opening and getting branch names can be significant
  - File byte-range caches take time to kick in, bad for small work packages



Fermilab

ROOT metadata is Awkward: large and complex structured

Typical ROOT file (CMS NanoAOD, DoubleMuon):

- 2.343 GB (13 GB uncompressed)
- 1405 TBranches; 413 for non-triggers (29%)
- 594 364 TBaskets; 252 496 for non-triggers (42%)
- 2.338 GB TBasket data; 3.6 kB/basket for triggers, 9.2 for non-triggers (88%)

ROOT metadata is Awkward: large and complex structured

Typical ROOT file (CMS NanoAOD, DoubleMuon):

- 2.343 GB (13 GB uncompressed)
- 1405 TBranches; 413 for non-triggers (29%)
- 594 364 TBaskets; 252 496 for non-triggers (42%)
- 2.338 GB TBasket data; 3.6 kB/basket for triggers, 9.2 for non-triggers (88%)

$\longrightarrow$ 14.7 MB as Awkward metadata; 4.9 MB gzip-compressed (ROOT uses 5.1 MB)

ROOT metadata is Awkward: large and complex structured

Typical ROOT file (CMS NanoAOD, DoubleMuon):

- 2.343 GB (13 GB uncompressed)
- 1405 TBranches; 413 for non-triggers (29%)
- 594 364 TBaskets; 252 496 for non-triggers (42%)
- 2.338 GB TBasket data; 3.6 kB/basket for triggers, 9.2 for non-triggers (88%)

$\longrightarrow$ 14.7 MB as Awkward metadata; 4.9 MB gzip-compressed (ROOT uses 5.1 MB)

$\longrightarrow$ uncompressed metadata is 0.6% as large as the data itself

ROOT metadata is Awkward: large and complex structured

Typical ROOT file (CMS NanoAOD, DoubleMuon):

- 2.343 GB (13 GB uncompressed)
- 1405 TBranches; 413 for non-triggers (29%)
- 594 364 TBaskets; 252 496 for non-triggers (42%)
- 2.338 GB TBasket data; 3.6 kB/basket for triggers, 9.2 for non-triggers (88%)

$\longrightarrow$ 14.7 MB as Awkward metadata; 4.9 MB gzip-compressed (ROOT uses 5.1 MB)

$\longrightarrow$ uncompressed metadata is 0.6% as large as the data itself

$\longrightarrow$ 6.7 TB dataset (2040 files) requires 41 GB of metadata in the database

ROOT metadata is Awkward: large and complex structured

Metadata as an Awkward Array (showing its type):

```
1 * {
    offsets: var * int64,
    era: var * {
        treename: string,
        names: var * string,
        interpretations: var * bytes    // pickle
    },
    prefix: var * string,
    file: var * {
        filename: string,
        tree: {
            run: var * {
                seek: int64,    // must be int64
                stop: int64,    // could be int32
                bytes: int64    // could be int32
            },
            luminosityBlock: var * {
                seek: int64,
                stop: int64,
                bytes: int64
            },
            event: var * {
                seek: int64,
                stop: int64,
                bytes: int64
            },
            CaloMET_phi: var * {
                seek: int64,
                stop: int64,
                bytes: int64
            },
            CaloMET_pt: var * {
                seek: int64,
                stop: int64,
                bytes: int64
            },
            ...         // 1405 TBranches
        }
    }
    era: int64,     // lookup era
    prefix: int64   // lookup prefix
}
```

Why is this structure useful?

- Global `offsets` to lookup files for a given range of entries.
- Small number of `eras` allows `treename`, TBranch `names`, and (pickled) `interpretations` to vary in the dataset (e.g. new trigger lines).
- `prefix` for `filename` avoids duplicated characters in long path URLs.
- Separate records for each TBranch allows TBranches to be queried independently. (TBranches that aren't in all files have `option` type.)
- Same number of TBasket `seek` positions, local entry `stop` values, and number of `bytes` share a variable-length list (**`var`**).

A perfect match for Tiled's slice-based predicate push-down

```
>>> awkward_client
<AwkwardClient>

>>> awkward_client[0, "offsets"]
<Array [0, 3271302] type='2 * int64'>   // just one file for now

>>> awkward_client[0, "prefix"]
<Array ['/home/jpivarski/storage/data/'] type='1 * string'>

>>> awkward_client[0, "file", ["filename", "prefix", "era"]][0].show()
{filename: 'Run2018D-DoubleMuon-Nano25Oct2019_ver2-v1-974F28EE-0FCE-4940-92B5-870859F880B
 prefix: 0,
 era: 0}

>>> awkward_client[0, "file", "tree", ["nMuon", "Muon_pt", "Muon_eta"]][0].show()
{nMuon: [{seek: 386614, stop: 1000, bytes: 587}, ..., {seek: 2511484157, ...}],
 Muon_pt: [{seek: 467837, stop: 1000, bytes: 8305}, ..., {seek: ..., ...}],
 Muon_eta: [{seek: 405661, stop: 1000, bytes: 6018}, ..., {seek: ..., ...}]}

>>> awkward_client[0, "file", "tree", 0, "nMuon", 100:103].show()
[{seek: 728247824, stop: 951800, bytes: 3223},
 {seek: 735505290, stop: 961318, bytes: 3243},
 {seek: 742659604, stop: 970836, bytes: 3195}]
```

Prototype: populate Tiled database, Uproot drop-in replacement

jpivarski / tiled-uproot Public

Pin | Unwatch 2 | Fork 0 | Star 1

main | 3 branches | 0 tags | Go to file | Add file | Code

jpivarski fix: some issues recognized in the practice before the Tiled talks (#3) ··· ✓ 02d5976 4 days ago 11 commits

.github	feat: extract arrays from Tiled-Uproot database (#2)	5 days ago
docs	initialize from scientific-python.org/cookie	3 weeks ago
src/tiled_uproot	fix: some issues recognized in the practice before the Tiled talks (#3)	4 days ago
tests	Populated lookup tables as an Awkward Array.	3 weeks ago
.git_archival.txt	initialize from scientific-python.org/cookie	3 weeks ago
.gitattributes	initialize from scientific-python.org/cookie	3 weeks ago
.gitignore	initialize from scientific-python.org/cookie	3 weeks ago
.pre-commit-config.yaml	Populated lookup tables as an Awkward Array.	3 weeks ago
.readthedocs.yml	initialize from scientific-python.org/cookie	3 weeks ago
LICENSE	initialize from scientific-python.org/cookie	3 weeks ago
README.md	initialize from scientific-python.org/cookie	3 weeks ago
noxfile.py	initialize from scientific-python.org/cookie	3 weeks ago
pyproject.toml	feat: extract arrays from Tiled-Uproot database (#2)	5 days ago

About

Stores ROOT metadata in Tiled for quicker sliced-array access.

Readme
BSD-3-Clause license
Activity
1 star
2 watching
0 forks

Releases

No releases published
Create a new release

Languages

Python 100.0%

After populating the database. . .

```
>>> tree = tiled_uproot.extract.TiledUproot("root_metadata", awkward_client)
>>> tree
<tiled_uproot.extract.TiledUproot object at 0x7fceb06c2d70>

>>> tree.show()
name                   | typename                 | interpretation
-----------------------+--------------------------+-------------------------------
run                    | uint32_t                 | AsDtype('>u4')
luminosityBlock        | uint32_t                 | AsDtype('>u4')
event                  | uint64_t                 | AsDtype('>u8')
CaloMET_phi            | float                    | AsDtype('>f4')
CaloMET_pt             | float                    | AsDtype('>f4')
CaloMET_sumEt          | float                    | AsDtype('>f4')
ChsMET_phi             | float                    | AsDtype('>f4')
ChsMET_pt              | float                    | AsDtype('>f4')
ChsMET_sumEt           | float                    | AsDtype('>f4')
nCorrT1METJet          | uint32_t                 | AsDtype('>u4')
CorrT1METJet_area      | float[]                  | AsJagged(AsDtype('>f4'))
CorrT1METJet_eta       | float[]                  | AsJagged(AsDtype('>f4'))
CorrT1METJet_muon...   | float[]                  | AsJagged(AsDtype('>f4'))
CorrT1METJet_phi       | float[]                  | AsJagged(AsDtype('>f4'))
CorrT1METJet_rawPt     | float[]                  | AsJagged(AsDtype('>f4'))
nElectron              | uint32_t                 | AsDtype('>u4')
```

After populating the database. . .

```
>>> array = tree.arrays(["nMuon", "Muon_pt"], entry_start=100, entry_stop=10000)
>>> array.show(type=True)
type: 9900 * {
    nMuon: uint32,
    Muon_pt: var * float32
}
[{nMuon: 2, Muon_pt: [10.4, 5.3]},
 {nMuon: 3, Muon_pt: [15.1, 9.15, 5.99]},
 {nMuon: 1, Muon_pt: [14.8]},
 {nMuon: 2, Muon_pt: [17.6, 12.9]},
 {nMuon: 0, Muon_pt: []},
 {nMuon: 2, Muon_pt: [49, 38.7]},
 {nMuon: 4, Muon_pt: [15.4, 6.3, ..., 5.15]},
 {nMuon: 3, Muon_pt: [15.5, 14.5, 12.8]},
 {nMuon: 1, Muon_pt: [20.1]},
 {nMuon: 1, Muon_pt: [9.4]},
 ...,
 {nMuon: 2, Muon_pt: [19.8, 12.2]},
 {nMuon: 1, Muon_pt: [3.14]},
 {nMuon: 1, Muon_pt: [13.1]},
 {nMuon: 1, Muon_pt: [19.6]},
 {nMuon: 2, Muon_pt: [17.4, 10.4]},
 {nMuon: 3, Muon_pt: [9.66, 4.06, 3.85]},
```

After populating the database. . .

```
>>> array = tree.arrays(
...     ["px", "py", "pz"],
...     cut="nMuon == 2",
...     aliases={
...         "px": "Muon_pt * cosh(Muon_eta) * cos(Muon_phi)",
...         "py": "Muon_pt * cosh(Muon_eta) * sin(Muon_phi)",
...         "pz": "Muon_pt * sinh(Muon_eta)",
...     },
...     entry_start=100,
...     entry_stop=10000,
... )
>>> array.show(type=True)
type: 4922 * {
    px: var * float32,
    py: var * float32,
    pz: var * float32
}
[{px: [-11.3, 1.18], py: [5.23, -5.32], pz: [6.89, ...]},
 {px: [14.9, 31], py: [10.3, 55.7], pz: [3.93, ...]},
 {px: [-46.3, 63.8], py: [-19.3, 7.58], pz: [10.5, ...]},
 {px: [16.3, -36.6], py: [51, -18.5], pz: [20.5, ...]},
 {px: [27.9, 3.71], py: [40.4, 18.8], pz: [-41.8, ...]},
 {px: [29.4, 15.1], py: [24.5, 5.01], pz: [-34.2, ...]},
```

Possible configuration: as an Adapter within Tiled