Awkward to RDataFrame and back

Ianna Osborne, Jim Pivarski

Princeton University, Princeton, NJ 08544, USA

E-mail: `ianna.osborne@cern.ch`

Abstract. Awkward Arrays [1] and RDataFrame [2] provide two very different ways of performing calculations at scale. By adding the ability to zero-copy convert between them, users get the best of both. It gives users a better flexibility in mixing different packages and languages in their analysis. In Awkward Array version 2, the `ak.to_rdataframe` function presents a view of an Awkward Array as an RDataFrame source. This view is generated on demand and the data are not copied. The column readers are generated based on the run-time type of the views. The readers are passed to a generated source derived from ROOT::RDF::RDataSource. The `ak.from_rdataframe` function converts the selected columns as native Awkward Arrays. The details of the implementation exploiting JIT techniques are discussed. The examples of analysis of data stored in Awkward Arrays via a high-level interface of an RDataFrame are presented. A few examples of the column definition, applying user-defined filters written in C++, and plotting or extracting the columnar data as Awkward Arrays are shown. Current limitations and future plans are discussed.

1. Introduction

Awkward Array is a library for nested, variable-sized data, including arbitrary-length lists, records, mixed types, and missing data, using NumPy-like [3] idioms.

In a typical Awkward user analysis workflow the access to columnar data is provided by ServiceX [4] and Uproot [5]. The data are presented to the user as Awkward Arrays. The user performs the event selection, applies the systematic uncertainties, produces histograms, builds a statistical model, likelihoods, statistical analysis, fit results and diagnostics. The user has access to a wide range of Python ecosystem packages - especially the Scikit-HEP packages [6] - to perform each task.

In contrast, the RDataFrame is a declarative, parallel framework for data analysis and manipulation. It reads from a columnar data format via a data source. It applies transformations to the data - that is, selects rows, defines new columns - and produces results. The results can be data reductions like histograms, new ROOT files, or any other user-defined object or side effect.

2. Awkward Array in Python eco-system

Awkward Arrays and RDataFrame are two very different ways of performing large-scale calculations. The `Awkward-RDataFrame` bridge provides users with more flexibility in mixing different packages and languages in their analyses if desired (see Fig. 1). There are numerous benefits of combining both Python and C++. The users can mix analyses using Awkward Arrays, Numba [8], and ROOT C++ in memory, without saving to disk and without leaving their environment.

On the one hand, the users who do their analysis entirely in Python eco-system can benefit from faster execution using ROOT C++ functions, or pure C++, in an otherwise Awkward analysis at full speed. The performance cost of converting Awkward Arrays into RDataFrame is negligible, since it is a zero-copy view, and the conversion of RDataFrame lists into Awkward Arrays is discussed below in section 4.

On the other hand, those who prefer the C++, ROOT, and RDataFrame, have an ability to convert their data into Awkward Arrays. This conversion opens many paths to follow.

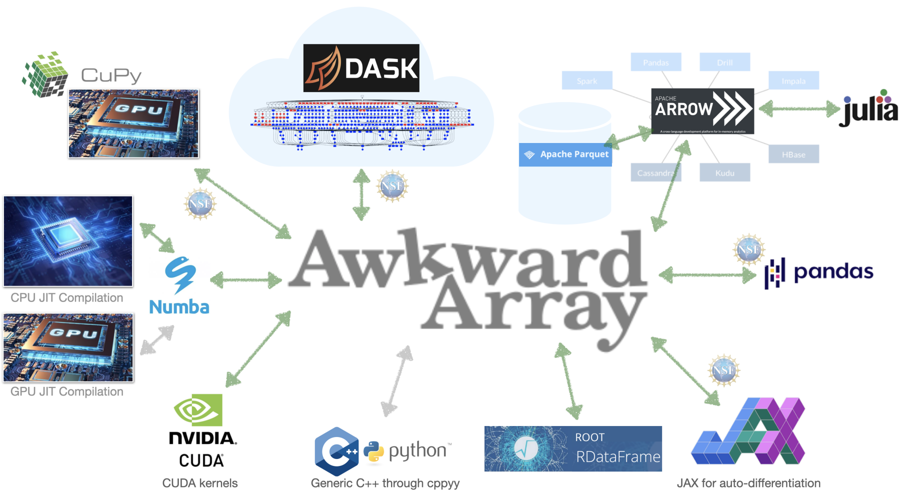

Figure 1. Awkward Array interoperability with other projects

3. From Awkward Array to RDataFrame

The Awkward-style `ak.to_rdataframe` function [9] presents a view of an Awkward Array as an RDataFrame source. The view is a lightweight 40-byte C++ object dynamically allocated on the stack. The generated RDataSource takes pointers into the original array data via this view. This view is generated on demand, the data are not copied - see figure 2. The view and the array data are accessible for as long as the RDataFrame returned by this function has not completed its execution and is currently in a running state.

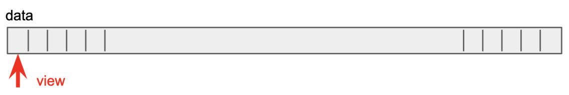

Figure 2. Awkward Array View can be thought as a cursor

This work takes advantage of an existing feature, in which Awkward Arrays can be iterated over in Numba-compiled functions, again using zero-copy views. The Numba implementation for C++ is being reused here: there is no performance difference. The column readers are generated based on the run-time type of the views. The readers are passed to a generated source derived from RDataSource.

The `ak.to_rdataframe` function takes a dict as its argument: each key defines a column name in the RDataFrame. The equal length arrays are given as values to the dictionary keys. The arrays data are not copied, but there is a small overhead of generating an Awkward RDataSource C++ code.

The following example describes three typical and distinct Awkward Arrays: an array of records, a flat array, and a list of arrays. Passing these arrays to RDataFrame creates three different columns with three different types.

The column `"x"` will maintain its awkward type, while the column `"y"` will become a column of integers, and the column `z` - a variable length array: each array is a container (`RVec`) of `double` values. The complete tutorial [11] including a toy analysis on CMS open data [12] has been presented at PyHEP2022 workshop.

```
import awkward as ak
import ROOT

array_x = ak.Array([
    {"x": [1.1, 1.2, 1.3]},
    {"x": [2.1, 2.2]},
    {"x": [3.1]},
    {"x": [4.1, 4.2, 4.3, 4.4]},
    {"x": [5.1]},])
array_y = ak.Array([1, 2, 3, 4, 5])
array_z = ak.Array([[1.1], [2.1, 2.3, 2.4], [3.1], [4.1, 4.2, 4.3], [5.1]])

assert len(array_x) == len(array_y) == len(array_z)

df = ak.to_rdataframe({"x": array_x, "y": array_y, "z": array_z})
```

This `ak.to_rdataframe` operation does not execute the RDataFrame event loop. The Awkward `RDataSource` is interpreted as Custom and the columns are its Datasets:

```
Dataframe from datasource Custom Datasource

Property                Value
--------                -----
Columns in total            4
Columns from defines        1
Event loops run             0
Processing slots            1

Column          Type                           Origin
------          ----                           ------
x               awkward::Record_DZ9qK2aXbBA    Dataset
y               int64_t                        Dataset
z               ROOT::VecOps::RVec<double>     Dataset
```

The RDataFrame column type is a string that corresponds to a C++ data type. The Awkward data types are defined in an `"awkward"` C++ namespace. Here, for example, the `"x"` column contains an Awkward Array with a made-up type: `awkward::Record_DZ9qK2aXbBA`. Awkward Arrays are dynamically typed and, in a C++ context, the type name is dynamically generated, and the name contains a hash of its contents to ensure uniqueness. In practice, there is no need to know the type. The C++ user code should use a placeholder type specifier `auto` - the type of the variable that is being declared will be automatically deduced from its initialiser.

The RDataFrame framework can perform all usual operations on Awkward data with one exception, the Snapshot operation. Presently, the Awkward type columns are not saved to a ROOT file. Scheduling an operation does not execute the event loop. For example, here is a filtering operation on all data where the column `"y"` values are greater than 2:

```
df = df.Filter("y % 2 == 0")
```

The filtered Awkward Array internal layout - a RecordArray data: its content NumpyArray - is not copied, it is indexed. It is wrapped in an IndexedArray - because of the filter selection. The other two columns data are copied. The same operation on Awkward arrays in Python produces the same result, the array of the same type, but a different internal layout - because Awkward arrays are immutable.

```
array_xyz = ak.Array({"x": array_x, "y": array_y, "z": array_z})
filtered_array = array_xyz[array_xyz["y"] % 2 == 0]
```

4. From RDataFrame to Awkward Array

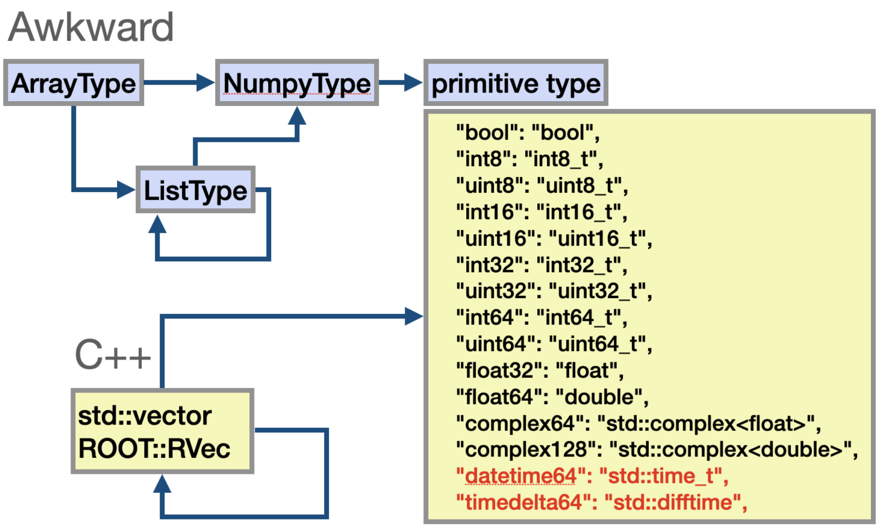

Figure 3. Supported Awkward Array and RDataFrame types. The last two lines under "primitive type" are red: they're not supported. An equivalent C++ type from the `std` C-style date and time utilities library hasn't been mapped to these Awkward types.

The `ak.from_rdataframe` function [10] converts selected columns to native Awkward Arrays. The function takes a string or a tuple of strings that are the RDataFrame column names and recognises the following column data types (see figure 3.):

- Primitive types: *integer*, *float*, *double*, $std :: complex < double >$, etc.
- Lists of primitive types and the arbitrary depth nested lists of primitive types: $std :: vector < double >$, $RVec < int >$, etc.
- Awkward types: the run-time generated types derived from $awkward :: ArrayView$ or $awkward :: RecordView$. There is no copy required because Awkward Arrays are immutable - a reference to the original input array is passed to the output.

The RDataFrame event loop is triggered once to retrieve all selected columns.

```
out = ak.from_rdataframe(df, columns=("x", "y", "z",), )
```

Both the C++ templated header-only Awkward-cpp implementation and the dynamically generated C++ code are used to extract the column types. This approach simplifies the JIT-compilation in ROOT. The array Python string description is constructed from the C++ data types. Both the dynamically generated from the Python string Layout builder [13] and the generated functions are needed to process the column data and are compiled with Cling [14].

5. Summary

Awkward Arrays and RDataFrame provide two very different ways of performing large scale calculations. By adding the ability to convert between them, users get the best of both. The Awkward-RDataFrame bridge provides users with more flexibility in mixing different packages and languages in their analyses. It is a part of Awkward version 2. The implementation is being adopted by RootInteractive project [15]. The user feedback is essential for further Awkward-RDataFrame development that is user-driven.

6. Acknowledgment

This work is supported by NSF cooperative agreement OAC-1836650 (IRIS-HEP) and NSF cooperative agreement PHY-2121686 (US-CMS LHC Ops).

References

[1] Pivarski, J., Osborne, I., Ifrim, I., Schreiner, H., Hollands, A., Biswas, A., Das, P., Roy Choudhury, S., Smith, N., Goyal, M. (2018). Awkward Array [Computer software]. `https://doi.org/10.5281/zenodo.4341376`

[2] RDataFrame: Easy Parallel ROOT Analysis at 100 Threads Danilo Piparo, Philippe Canal, Enrico Guiraud, Xavier Valls Pla, Gerardo Ganis, Guilherme Amadio, Axel Naumann, Enric Tejedor EPJ Web Conf. 214 06029 (2019) `DOI: 10.1051/epjconf/201921406029`

[3] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). `DOI: 10.1038/s41586-020-2649-2`

[4] Galewsky, B., Gardner, R., Gray, L., Neubauer, M., Pivarski, J., Proffitt, M., Vukotic, I., Watts, G., Weinberg, M., ServiceX A Distributed, Caching, Columnar Data Delivery Service, EPJ Web Conf. 245 04043 (2020) `DOI: 10.1051/epjconf/202024504043`

[5] Pivarski, J., Schreiner, H., Hollands, A., Das, P., Kothari, K., Roy, A., Ling, J., Smith, N., Burr, C., Stark, G. (2017). Uproot [Computer software]. `https://doi.org/10.5281/zenodo.4340632`

[6] Scikit-HEP project `https://scikit-hep.org`

[7] Rene Brun, Fons Rademakers, Philippe Canal, Axel Naumann, Olivier Couet, Lorenzo Moneta, Vassil Vassilev, Sergey Linev, Danilo Piparo, Gerardo GANIS, Bertrand Bellenot, Enrico Guiraud, Guilherme Amadio, wverkerke, Pere Mato, TimurP, Matevž Tadel, wlav, Enric Tejedor, ... Raphael Isemann. (2019). root-project/root: v6.18/02 (v6-18-02). Zenodo. `https://doi.org/10.5281/zenodo.3895860`

[8] Siu Kwan Lam, stuartarchibald, Antoine Pitrou, Mark Florisson, Stan Seibert, Graham Markall, esc, Todd A. Anderson, rjenc29, Guilherme Leobas, luk-f-a, Michael Collison, Jay Bourque, Aaron Meurer, Travis E. Oliphant, Nick Riasanovsky, Kaustubh, Michael Wang, densmirn, ... James Bourbeau. (2022). numba/numba: Version 0.56.4 (0.56.4). Zenodo. `https://doi.org/10.5281/zenodo.7289231`

[9] `https://awkward-array.org/doc/main/reference/generated/ak.to_rdataframe.html`

[10] `https://awkward-array.org/doc/main/reference/generated/ak.from_rdataframe.html`

[11] Osborne, I., Pivarski, J., (2022, September 15). Awkward RDataFrame Tutorial. PyHEP 2022 (virtual) Workshop. Zenodo. `https://doi.org/10.5281/zenodo.7081586`

[12] CERN Open Data portal `DOI:10.7483/OPENDATA.CMS.LVG5.QT81` (CMS data)

[13] Goyal, M., Osborne, I., Pivarski, J. The Awkward world of Python and C++, these proceedings

[14] Cling [Online]. `https://root.cern.ch/cling`

[15] ROOTInteractive `https://github.com/miranov25/RootInteractive`