ALICE feedback on RNTuple

Giulio Eulisse

ALICE Run 3 Analysis Framework recap

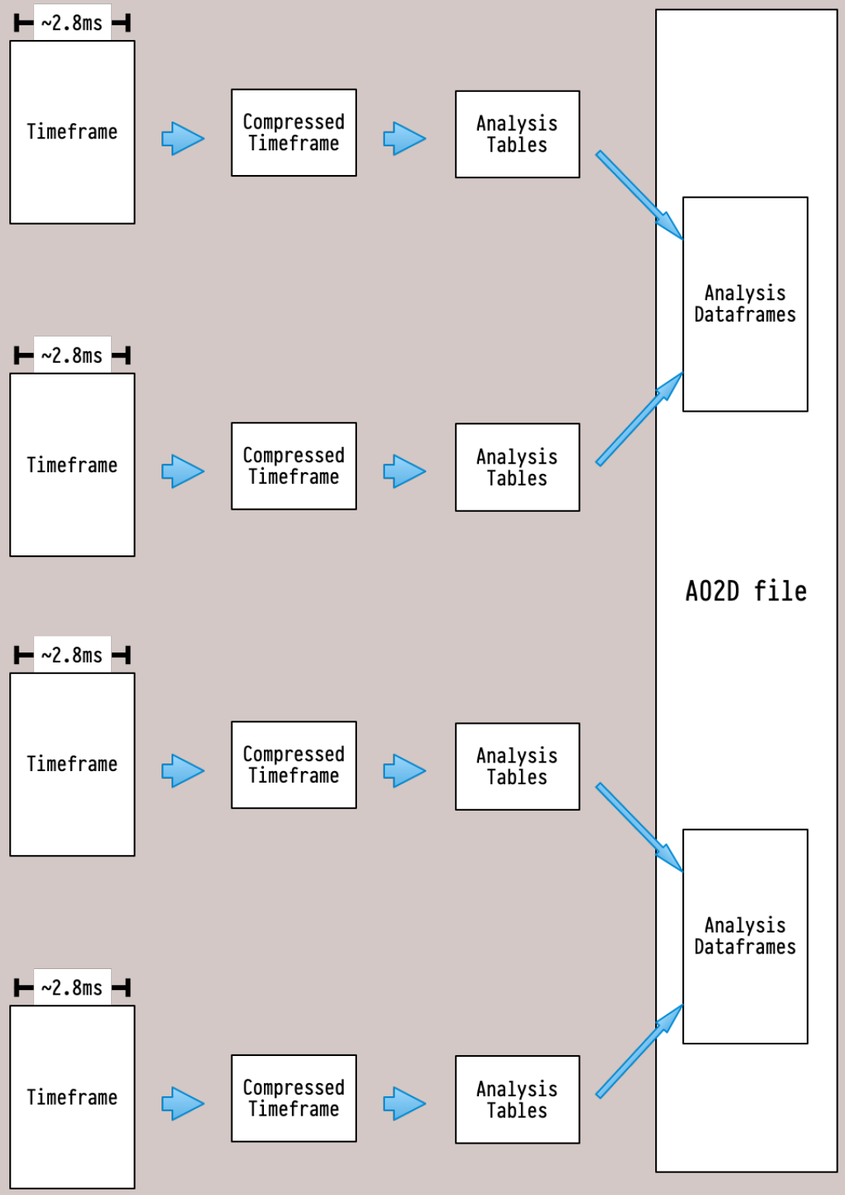
Timeframe: ~2.3 ms continuous readout detector data. Custom in-memory format.

Compressed Timeframes: result of reconstruction. ROOT based custom format.

Analysis Tables: tabular representation of analysis related quantities (e.g. tracks, collisions, calorimetry information, triggers).

Dataframe: concatenated analysis tables extracted from CTFs. Stored as ROOT `TTree`s in a `TFolder`. O(100MB).

AO2D file: ROOT `TFile` containing O(100) Dataframes.



ALICE Analysis Data Model

Relational DB-like. Analysis quantities are described in terms of tables, made of columns, and their relationships.

Simple: there are only a few dozens of tables produced by reconstruction (i.e. excluding derived user data).

Schema definition. C++ macros are used to define columns and group them into tables. Basic types are supported and some complex ones (e.g. fixed size arrays and VLAs).

ORM layer. The macros also build an Object Relational Mapping layer which provides the user the familiar `track.pt()` experience.

Grouping is dynamic and part of the query (i.e. the C++ analysis task), not part of the data model.

Some special **"dynamic columns"** are computed in batch at read time via Arrow's Gandiva expression compiler.

```
namespace collision
{
DECLARE_SOA_INDEX_COLUMN(BC, bc);                              //! Most probably BC
DECLARE_SOA_COLUMN(PosX, posX, float);                         //! X Vertex position in cm
DECLARE_SOA_COLUMN(PosY, posY, float);                         //! Y Vertex position in cm
DECLARE_SOA_COLUMN(PosZ, posZ, float);                         //! Z Vertex position in cm
DECLARE_SOA_COLUMN(CovXX, covXX, float);                       //! Vertex covariance matrix
DECLARE_SOA_COLUMN(CovXY, covXY, float);                       //! Vertex covariance matrix
DECLARE_SOA_COLUMN(CovXZ, covXZ, float);                       //! Vertex covariance matrix
DECLARE_SOA_COLUMN(CovYY, covYY, float);                       //! Vertex covariance matrix
DECLARE_SOA_COLUMN(CovYZ, covYZ, float);                       //! Vertex covariance matrix
DECLARE_SOA_COLUMN(CovZZ, covZZ, float);                       //! Vertex covariance matrix
DECLARE_SOA_COLUMN(Flags, flags, uint16_t);                    //! Run 3: see Vertex::Flags
DECLARE_SOA_COLUMN(Chi2, chi2, float);                         //! Chi2 of vertex fit
DECLARE_SOA_COLUMN(NumContrib, numContrib, uint16_t);          //! Number of tracks used for the vertex
DECLARE_SOA_COLUMN(CollisionTime, collisionTime, float);       //! Collision time in ns relative to BC
DECLARE_SOA_COLUMN(CollisionTimeRes, collisionTimeRes, float); //! Resolution of collision time
} // namespace collision

DECLARE_SOA_TABLE(Collisions_000, "AOD", "COLLISION", //! Time and vertex information of collision
                  o2::soa::Index<>, collision::BCId,
                  collision::PosX, collision::PosY, collision::PosZ,
                  collision::CovXX, collision::CovXY, collision::CovXZ,
                  collision::CovYY, collision::CovYZ, collision::CovZZ,
                  collision::Flags, collision::Chi2, collision::NumContrib,
                  collision::CollisionTime, collision::CollisionTimeRes);
```

I/O

Fully ROOT based:

- **AO2D files** are `TFile`s.
- **Each table** is a `TTree`s.
- **Dataframes** are just trees in `TFolder`s.
- **Some small metadata** present in terms of key - value pairs stored in a `TMap`.

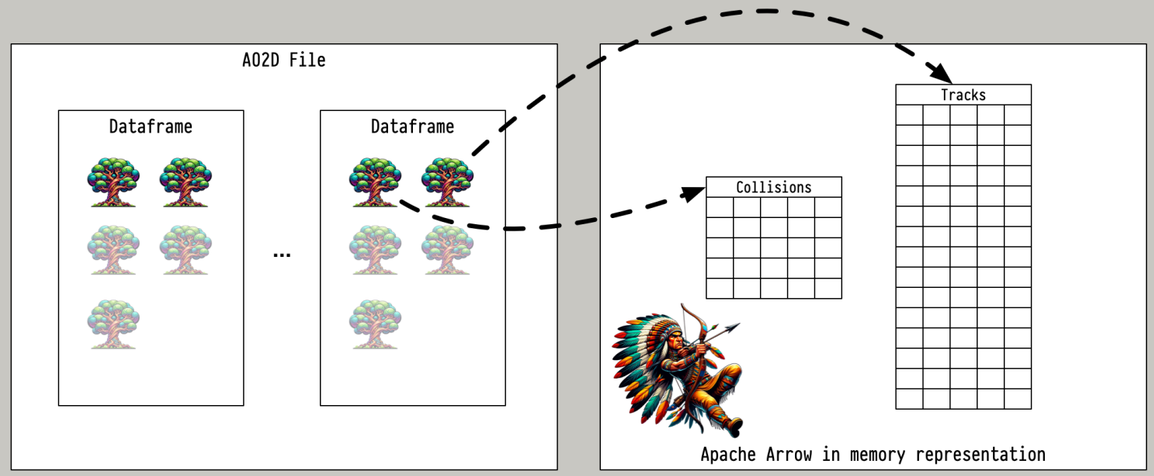
The **unit of reading** is the data frame table (e.g. tracks) which is loaded on demand.

Tables / trees are loaded in shared memory as **Apache Arrow** tables. The ORM layer works directly on top of Arrow, not on top of some C++ object graph.

TBulkBranchRead is used to do the actual read. While rather performant, it unfortunately requires one extra copy to shared memory.

Least significant bits zero-ing to improve compression of certain column, physics permitting.

Schema evolution is handled via special conversion tasks.



RNTuple ALICE interests

Different usecase. We are rather happy with the current setup. Admittedly it is a rather different design compared to others. We would however profit a lot from a **clean and performant API for reading / writing columns of data** to storage.

Simple usecase: no `TRef`s, no `std::kitchen_sink`, no polymorphism, no schema evolution.

Hidden from the user. Baseline is that users will keep using our ORM layer.

Prototype. All the investigations done so far are limited proofs of concept. Converting our data to RNTuple does not give much gains in terms of file size, as expected. We are still evaluating the reading performance part.

Avoid one copy. Our major performance issue with `TBulkBranchRead` at the moment is the extra copy it forces on us. We would greatly appreciate some Bulk API which allows us to decompress directly to our own Arrow `Table`s / `RecordBatch`es.

Per-column compression algorithms. E.g. indices would benefit from delta encoding.

Current issues

Documentation is a tad sparse at the moment. In particular one would benefit from a top-level view.

Issue with `TFolder`s: RNTupleWriter does not seem to know about TFolders. See #14007.

Bulk API:

- **Not in v6.28.04**. ALICE SW for now does not work with 6.30.00-rc1.
- **Writing**. Not there, AFAICT.
- **Ownership**. It's still impossible to have the bulk API write to non-owned buffers.

A lot of people like **TTree::Draw**.

Analysis happens centrally in Analysis Facilities / Grid (Trains!). Any improvement from special features needs to be deployed there for ALICE to exploit.

Kudos...

... to Jakob for the help...