A Large Ion Collider Experiment

Recent ALICE user experience / wishes for ROOT-IO

Sandro Wenzel

Outline

Presenting a collection of things we have come across recently ...

1. **“Difficulties” with ROOT-IO in the context of RUN3 developments (ALICE O2 project)**
2. **An improved TClonesArray**
3. **IO feature request motivated from “ObjCmp” - a tool to compare ROOT serialized objects**

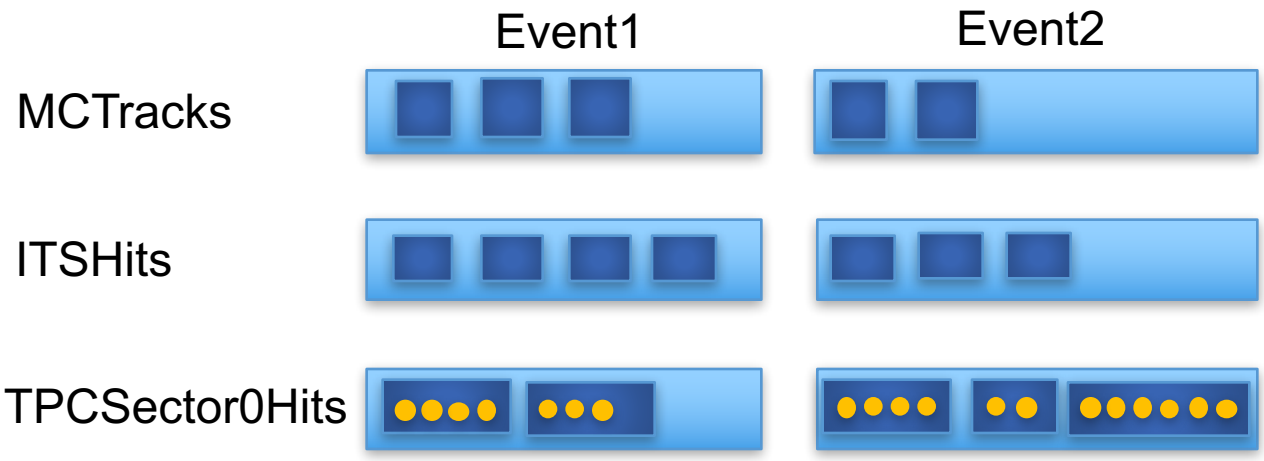
RECENT USER EXPERIENCE WITH ROOT IO

- Our experience with ROOT IO is very good in general!
- **“Good”** can definitely be considered the general **“background” …**
- Some unexpected **“issues”** emerged as the **“signal”** on which I would like to report …
  - Some of the things mentioned probably a misuse of ROOT on our side

Context

Data organisation/model for simulation changes from Run2 (AliRoot) to Run3 (based on FairRoot)

- Will store (sim) data objects in branches, with entries corresponding to events or continous time ranges
- Each entry is a TClonesArray!
- Elements of the TClonesArray can have other containers
- Entries can be **very large (TPC)**

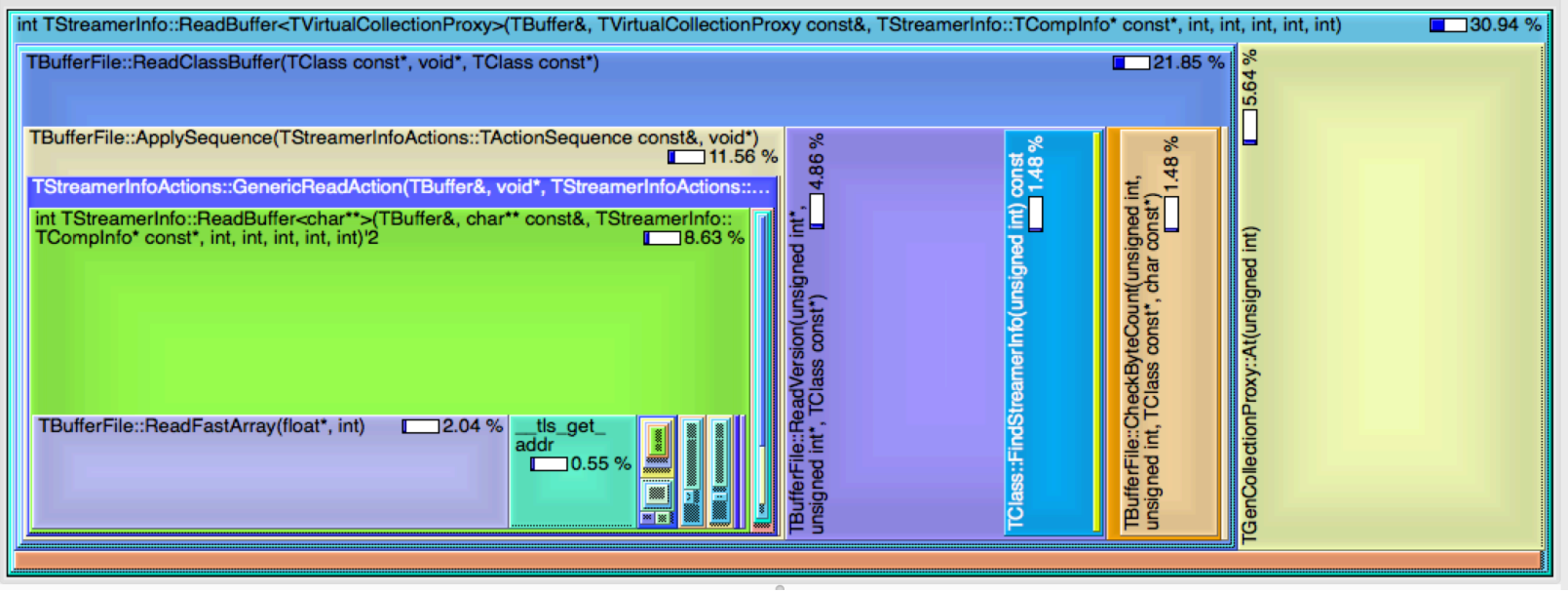


Branch entry = TClonesArray of simulation data objects

Element of TClonesArray
- Must derive from TObject
- Usually corresponds to one data element (hit)

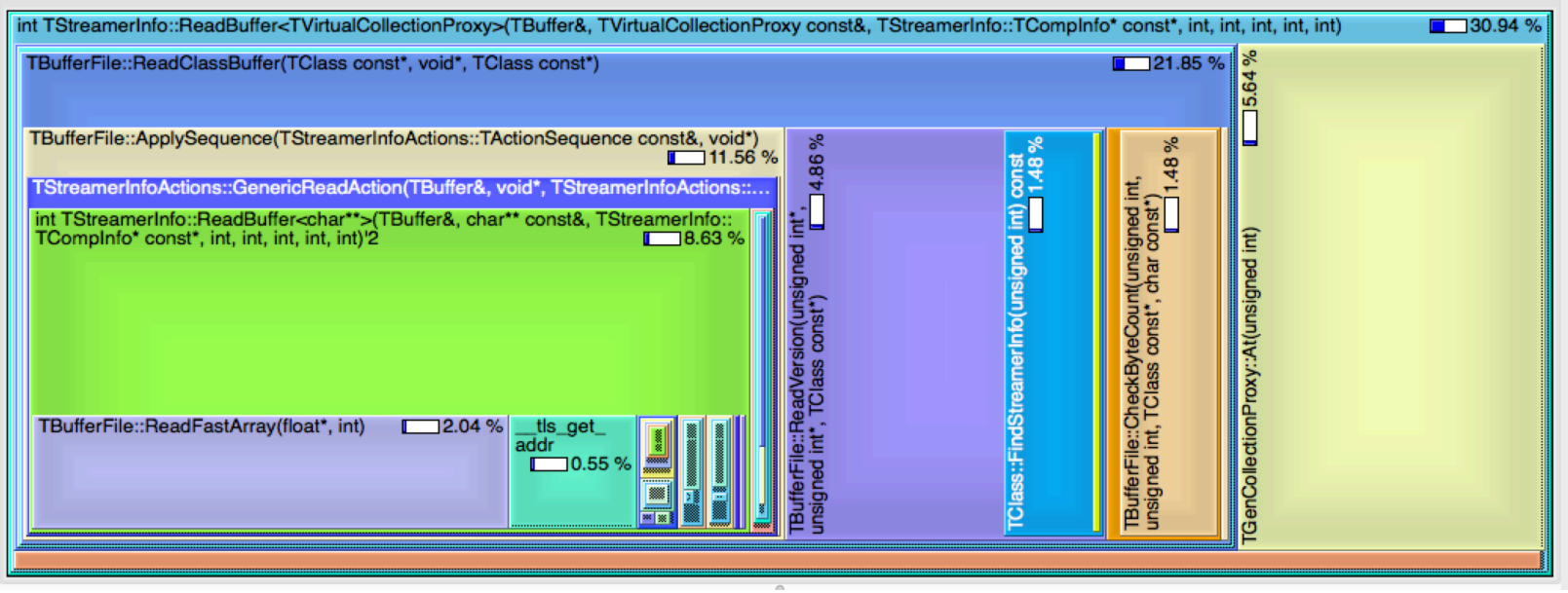
Element of TClonesArray as **wrapper to a container of elemental things**
- Group TPC hits per track+sector;
- The elemental things do not need to derive from TObject

Concern 1: IO of std::vector and co?



- We use, e.g., std::vector containers inside objects (in a TClonesArray)
  - `std::vector<Hit>`
- Observed that IO takes considerable CPU, caused by iterating over the vector using TGenCollectionProxy + virtual functions
- Effect vanishes using
  - `Hit *mHits; //[Size]`

Concern 1: IO of std::vector and co?



- We use, e.g., std::vector containers inside objects (in a TClonesArray)
  - `std::vector<Hit>`
- Observed that IO takes considerable CPU, caused by iterating over the vector using TGenCollectionProxy + virtual functions
- Effect vanishes using
  - `Hit *mHits; //[Size]`
- Feature request: Fast IO! In particular, of course, that “IO of std::containers should be as fast as the C-variants”

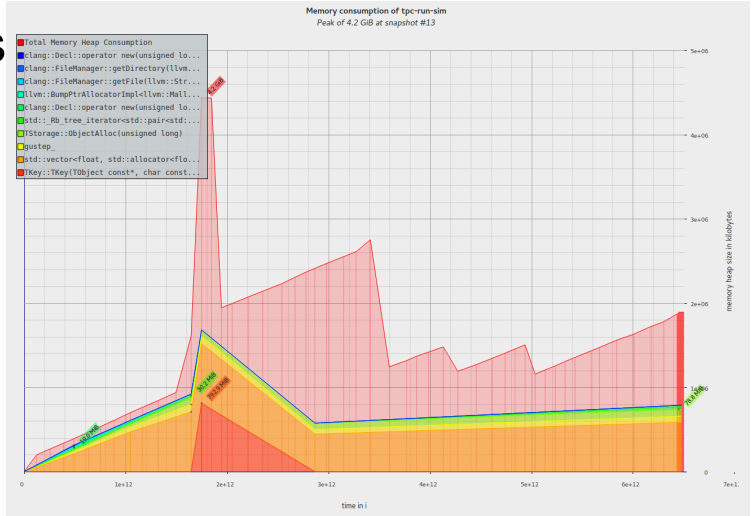
Concern 2: O(N^2) scaling while TTree::Drawing()

- Observed a very slow response with TTree::Draw() in a context, where
  - Have **many data** objects in a TClonesArray
  - Each individual data object holds a **small variable** sized vector `std::vector<int> mMCLabels`
- Drawing on `mMCLabels` takes “infinitely look”
  - The calculation of the right index (happening in TTreeFormula) to access the data suffers from a scaling problem
  - Probably only visible in certain circumstances
- Problem already discussed with P. Canal. A pull request fixing the scaling problem will be available soon.

Concern 3: Memory hoarding bug and testing

- Observed a memory “hoarding” problem, where ROOT IO triggered a **4x multiplication of the allocated memory**, essentially preventing us from simulating very large events

- Now fixed in v6-08-patches and others



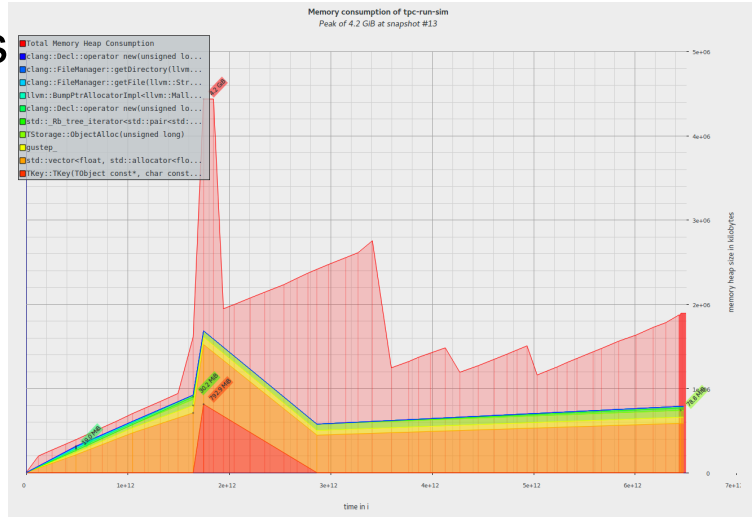
Concern 3: Memory hoarding bugs and testing

- Observed a memory “hoarding” problem, where ROOT IO triggered a **4x multiplication of the allocated memory**, essentially preventing us from simulating very large events

- Now fixed in v6-08-patches and others

- **What can we learn from these lessons?**

- **Importance to enhance testing!**
  - Enlarged test suites for IO
  - Monitor CPU + memory consumption for typical use cases and install alarm
  - Backport important fixes to all supported releases



Feature request on “TClonesArrays”

Evolution of TClonesArray

- TClonesArray is a nice feature of ROOT
  - Memory managed container, with particular strengths for large objects
  - Nice ROOT IO integration (constructor/deconstructor skipping; object reuse) as the basis for good speed and user convenience
  - Used a lot in FairRoot (i.e., the “Absorb” functionality) and currently central to our simulation data model

Evolution of TClonesArray

- TClonesArray is a nice feature of ROOT
  - Memory managed container, with particular strengths for large objects
  - Nice ROOT IO integration (constructor/deconstructor skipping; object reuse) as the basis for good speed and user convenience
  - Used a lot in FairRoot (i.e., the “Absorb” functionality) and currently central to our simulation data model
- But:
  - **not type safe** … (awkward code)
  - requires elements to be a **TObject**, which can lead to considerable memory overhead
  - somewhat weird API (with unexpected side-effects) and rather ancient look and feel
  - **sorting not competitive with stl** (virtual functions vs templates)

Evolution of TClonesArray (cont)

- Given the important role this container plays for us, we would be interested having a modernized version keeping all the benefits while addressing the current shortcomings
  - type awareness (templated); “no TObjects”; stl like; …; Same good ROOT IO integration !!
- Some work was already invested on this and a first prototype is available
  - TManagedVector
- Hope to get more people interested and to advance the discussions

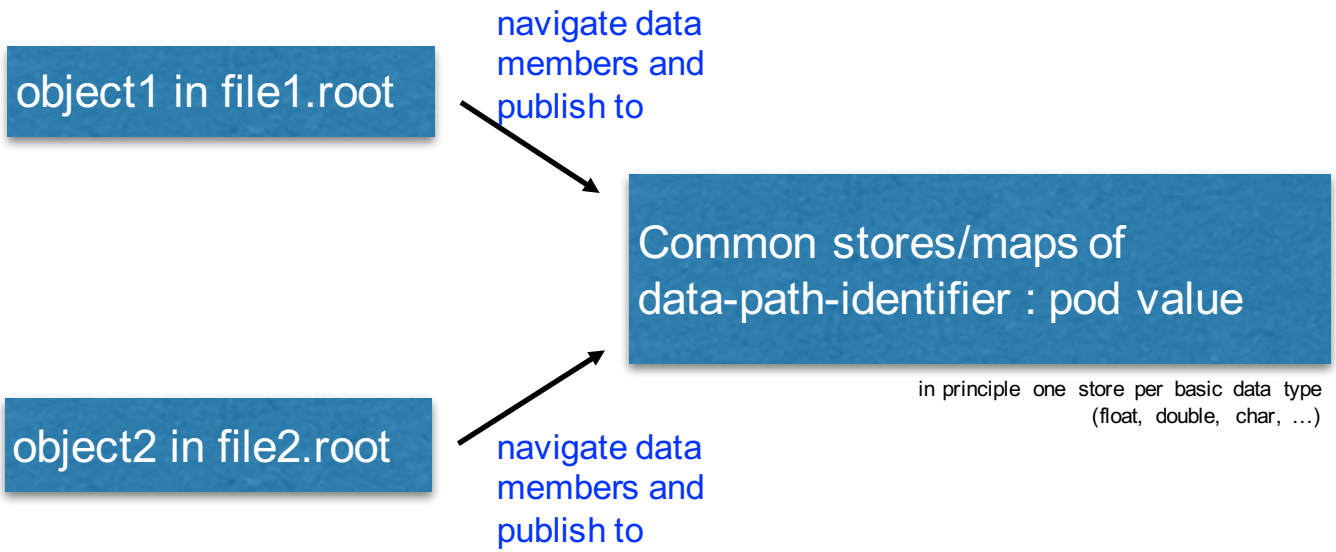
IO feature request motivated from “ObjCmp”

Motivation

- Most of our data is encapsulated in C++ objects and serialised to disc using ROOT I/O …
- We want to test for bitwise compatibility of objects sitting in different root files …
  - **MUCH** more complicated than comparing the file in binary mode (because of timestamps and other information that ROOT adds)!!
- We want to obtain a measure how much objects in different root files differ…

Idea using introspection

- Use **introspection** capabilities of ROOT in order to loop and hierarchically navigate through data members of C++ objects
  - publish encountered (leaf) data to in-memory stores/map(s)
  - use this data-store to easily retrieve differences



How it looks like : Simple Printing

```
struct Point {
  Point() = default;
  Point(double x, double y) : mX(x), mY(y) {}
  double mX = -110.;
  double mY = 10;
};

struct Track {
  int mId = -1;
  std::vector<Point> mPoints{Point(-1,10), Point(2,10)};
};
```

```
"PrintObject ../test/Track.root Track"

        Track.mId{int}  : -1
        Track.mPoints{vector<Point>}[0].mX{double}  : -1
        Track.mPoints{vector<Point>}[0].mY{double}  : 10
        Track.mPoints{vector<Point>}[1].mX{double}  : 2
        Track.mPoints{vector<Point>}[1].mY{double}  : 10
```

unique data identifier

pod value

OCDB diff

```
.CDBCompare Run244918_244918_v2_s0.root Run244918_244918_v3_s0.root
```

(/alice/data/2015/OCDB/TOF/Calib/RunParams/)

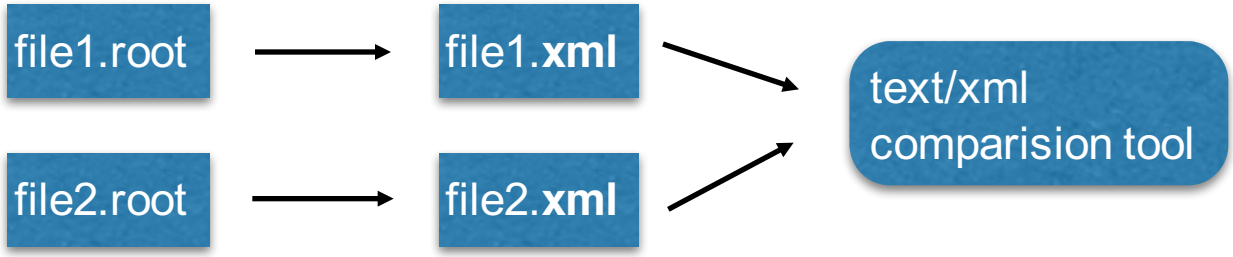
```
DIFFERENCE FOR KEY AliCDBEntry.fId{AliCDBId}.fVersion{int} ABSOLUTE -1

//…

DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fT0{float*}[0] ABSOLUTE -7.72662
DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fT0{float*}[1] ABSOLUTE -3.46628
DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fT0{float*}[2] ABSOLUTE -2.82877

//…

DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fTOFResolution{float*}[0] ABSOLUTE 26.0465
DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fTOFResolution{float*}[1] ABSOLUTE 30.7094
DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fTOFResolution{float*}[2] ABSOLUTE 26.1067

//…

DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fT0Spread{float*}[19] ABSOLUTE -0.0553589
DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fT0Spread{float*}[20] ABSOLUTE 1.43106
DIFFERENCE FOR KEY AliCDBEntry.fObject{AliTOFRunParams}.fT0Spread{float*}[21] ABSOLUTE 1.00253

//…
```

ObjCmp: Feature Request

- Currently **required to have the dictionary libs** available at the moment of reading/navigation C++ objects …. although the complete class structure is also part of the serialized data in form of **TStreamerInfos**
- An attempt to implement ObjCmp purely based on TStreamerInfos failed due to a few unsupported things
- **“As a user I would like to be able to fully navigate serialized ROOT objects without needing the dictionaries used to serialize them.”**
- By-product: Could increase user experience also for navigating objects in the TBrowser
- ”Should object comparison be native to ROOT?”

BACKUP

Comparing objects based on text tools



- Pro:
  - simple “diff” might be a good enough in simple cases
- Contra:
  - multi stage process
  - intermediate text output size can be huge
  - still requires external tools (xml parser) to parse and extract differences between 2 xml files
  - text handling is slow
  - loss of precision for double/float values
  - text encoding might be difficult to interpret and diff (for example compression scheme for std::bitset…)