EPJ Web of Conferences **245**, 06032 (2020) https://doi.org/10.1051/epjconf/202024506032
CHEP 2019

Data Analysis using ALICE Run 3 Framework

Giulio Eulisse[1,*], *Anton* Alkin[2], *Jan Fiete* Grosse-Oetringhaus[1], *Peter* Hristov[1], *Gian Michele* Innocenti[1], and *Maja Jadwiga* Kabus[3]

[1]Organisation Européenne pour la Recherche Nucléaire, CERN, Meyrin, Switzerland
[2]Bogolyubov Institute for Theoretical Physics of the National Academy of Sciences of Ukraine, Kyiv, Ukraine
[3]Warsaw University of Technology, Warsaw, Poland

Abstract. The ALICE Experiment is currently undergoing a major upgrade program, both in terms of hardware and software, to prepare for the LHC Run 3. A new Software Framework is being developed in collaboration with the FAIR experiments at GSI to cope with the 100-fold increase in the number of recorded events. We present our progress to adapt such a framework for the end user physics data analysis. In particular, the design and technology choices are highlighted. How Apache Arrow is adopted as the platform for the in-memory analysis data layout is discussed. The benefits of this solution are illustrated, these include: efficient and parallel data processing, interoperability with a large number of analysis tools and ecosystems, integration with the modern ROOT declarative analysis framework RDataFrame.

1 ALICE analysis in Run 1 and 2

The implications for the software architecture and framework of the ALICE experiment upgrades [1] for the so called Run 3 of the Large Hadron Collider (LHC) have already been detailed elsewhere, in particular describing the new Online - Offline (O^2 [2]) architecture and its Data Processing Layer (DPL [3]). How those changes will reflect on the end-user analysis software and how it is run is hereby illustrated.

During Run 1 and Run 2, ALICE physicists were able to run their analysis using two kinds of physics objects:

- Event Summary Data (ESD), i.e. the detailed reconstruction output, including multiple reconstruction snapshots for calibration and QA purposes;
- Analysis Object Data (AOD), i.e. an analysis specific, distilled subset of the previous.

In both cases, the event was represented as an object, stored using ROOT [4] serialization capabilities.

In order to process data, physicists wrote analysis tasks, nicknamed wagons, organized in workflows, nicknamed trains, which run on the data using the WLCG Grid via ALICE developed Grid submission Middleware, AliEn [5]. This allowed the experiment to offset the cost of data access per wagon, since data is read once per train.

*e-mail: giulio.eulisse@cern.ch

While the core of such an architecture will remain, the fact that ALICE will take hundred times more (minimum bias) collisions in Run 3, necessitates significant changes for both the computing model and the software to cope with the increased data rate. In particular, it will not be feasible to store and retrieve the equivalent of ESD during the Run, so all the analyses will have to be performed at AOD level. AODs will not contain any quantities that can be calculated on the fly, in order to fit into the estimated 100-fold increase in required throughput.

2 ALICE computing model in Run 3

A first challenge is to reduce the latency of Analysis Trains while tuning the various aspects of analysis. In order to achieve this, an extension to our computing model to include a new entity named "Analysis Facility" is planned. This is an optimized computing resource, hosting 10% of the data, which will be used to run all the trains daily and provide a quick feedback loop. The plan is that only the wagons or trains which qualify will then be allowed to run on the whole dataset, over a few days, by running jobs on the Grid.

A second aspect that concerns the computing model is the fact that certain well defined analysis on rare processes, like for instance certain heavy-flavoured probes, high p_{T} jets, and nuclei, will be allowed to store organized selections (skims) of data, so that the amount of data to be read for them will be highly reduced [6]. In the past, the advantages this approach were overweighted by the bookkeeping concerns, however the increased data rates of Run 3 is expected to move the balance in favour of a well maintained set of skimmed datasets.

Finally, lossy compression of the data by zeroing the least significant bits of the mantissa of floating point will be performed, as allowed by the uncertainty on the represented quantity.

The goal, as stated in the ALICE Software and Computing TDR [2], is to go through the equivalent of 5 PB of AODs every 12 hours (100 GB/s) which roughly translates to 20 MB/s/-core with the current resource estimates for the Analysis Facilities.

3 ALICE analysis framework in Run 3

The analysis software of the experiment will also undergo a significant reorganization, aimed at improving its performance and integrating it into the new O^2 Data Processing Layer (DPL) in order to provide a coherent environment from data taking to analysis.

The same terminology of Run 2, where wagons (tasks) will be organized in trains (workflows) will be maintained. However, each wagon will be mapped on a DPL Data Processor. This will allow us to parallelise the processing of our wagons, both in terms of parallel execution on a single timeframe, and in terms of ability to pipeline the processing of multiple timeframes. Moreover, the inherently distributed and dynamic nature of the ALICE O^2 framework opens the possibility to run on multi-node slots (e.g. to amortize the cost of I/O and common computations) or to remove poorly performing or crashing wagons.

Compared to the current framework, the underlying data model will be flattened out in set tables arranged in a relational-database-like manner. The goal is to minimize the cost due to serialization, exploit the shared memory backend of FairMQ, and to pave the way for vectorised processing of the bulk data present in the timeframe.

As a baseline, tables will be stored as a set of flat ROOT trees, both for original AODs and for derived skimmed data. Histograms and output objects in general will be serialized using ROOT objects serialization to facilitate drawing.

In order to minimize the amount of changes that physicists will have to do, a compatibility API which will allow the users to port their old code incrementally, with minimal adjustments,

will be provided. However, for the analyses which are on the critical path a more declarative API will be recommended, as it allows for the optimization and reuse of common filters and computations.

3.1 Core features of the analysis framework

As previously said, data is described in terms of Tables. Each table is merely the union of columns and some metadata associated to it. In order to define the schema for a given table, users have to specify the columns, via the `DECLARE_SOA_COLUMN` macro and group them into tables via the `DECLARE_SOA_TABLE` macro (see listing 1 for an example). This approach is not dissimilar to those being investigated by the CMS [7] and LHCb [8] collaborations, with the notable difference that the backing store for the columns will be provided by the open-source library Apache Arrow (Arrow, [9]).

```
namespace track
{
// An index column, referring to the separate Collision table.
DECLARE_SOA_INDEX_COLUMN(Collision, collision);
// The declaration of a normal column, associated to the C++ type track::X,
// with accessor X::x() and storing a float in the fX branch.
DECLARE_SOA_COLUMN(X, x, float, "fX");
DECLARE_SOA_COLUMN(Alpha, alpha, float, "fAlpha");
DECLARE_SOA_COLUMN(Y, y, float, "fY");
DECLARE_SOA_COLUMN(Z, z, float, "fZ");
DECLARE_SOA_COLUMN(Snp, snp, float, "fSnp");
DECLARE_SOA_COLUMN(Tgl, tgl, float, "fTgl");
DECLARE_SOA_COLUMN(Signed1Pt, signed1Pt, float, "fSigned1Pt");
// A dynamic column, i.e. a column which is calculated on the fly from the contents of other columns
DECLARE_SOA_DYNAMIC_COLUMN(Pt, pt, [](float signed1Pt) -> float { return fabs(1.0 / signed1Pt); });
DECLARE_SOA_DYNAMIC_COLUMN(Phi, phi, [](float snp, float alpha) -> float { return asin(snp) + alpha
     + M_PI; });
DECLARE_SOA_DYNAMIC_COLUMN(Eta, eta, [](float tgl) -> float { return log(tan(0.25 * M_PI - 0.5 *
     atan(tgl))); });

}

// The declaration of the Tracks table, mapping on the O2 Message with header
// AOD/TRACKPAR and consisting of the specified columns.
DECLARE_SOA_TABLE(Tracks, "AOD", "TRACKPAR",
                  o2::soa::Index<>, track::CollisionId, track::X, track::Alpha,
                  track::Y, track::Z, track::Snp, track::Tgl,
                  track::Signed1Pt,
                  track::Phi<track::Snp, track::Alpha>,
                  track::Eta<track::Tgl>,
                  track::Pt<track::Signed1Pt>,
                  track::Charge<track::Signed1Pt>);

// Shortcut to iterate over the rows of the Tracks table.
using Track = Tracks::iterator;
```

Listing 1. An example for the schema of table with the track propagation properties.

Besides standard columns, which correspond to a branch in a ROOT tree, it is also possible to define as columns indices to other tables or quantities to be calculated on the fly, either when requested, or in bulk at the beginning of the first requesting task. These quantities are planned to be computed only once for complex operations and then cached for all tasks in the workflow.

The table declaration will effectively describe an Apache Arrow (Arrow) table which will be filled by the values in a TTree on disk or computed by a given task. Exploiting Apache Arrow gives us solid foundations for the in-memory layout of the data, and fits particularly well the zero-copy, shared memory backed, message passing paradigm of the O^2 framework.

While only basic types are supported at the moment for columns, we plan to expose, at least partially some of the nested types provided by Arrow, e.g. to handle jet constituents.

In order to process the data, the user has to provide a task struct with (at least) a `process()` method. Its signature will define to which tables a given task subscribes. Certain special members of the struct will allow the user to describe Filters on the data, new tables produced by the task and configurable parameters associated to the task (see listing 2 for an example).

```
struct SomeTask {
  OutputObj<TH2F> etaphiH{TH2F("etaphi", "etaphi", 100, 0., 2. * M_PI, 100, -2.f, 2.f)};
  // Create a configurable which can be used inside the process method.
  Configurable<float> ptCut{"pt", 0.1f, "A cut on phi"};

  void process(aod::Tracks const& tracks)
  {
    for (auto& track : tracks) {
      if (track.pt() > ptCut) {
        etaphiH->Fill(track.phi(), track.eta());
      }
    }
  }
};
```

Listing 2. An example task which subscribes to the track table and produces an histogram with the η and ϕ of all the processed tasks.

The framework inspects the structure of a wagon and creates the associated DPL DataProcessor which will describe the computation. In case it is desirable from a performance point of view, the framework might decide to stack multiple wagons in a single data processor if the data flow allows for it.

Given the importance that selection and filtering has in a physics analysis, the vectorized query engine provided by the Apache Arrow subproject Gandiva [10] has been integrated in the framework. Such an engine allows us to create a declarative C++17 based Domain Specific Language which allows the physicist to write filters on the table columns in a declarative way. The framework will then convert such a description in an Abstract Syntax Tree, which is then used to Just-In-Time generate the actual code for the query, vectorised as needed, thanks to the LLVM compiler infrastructure [11]. A simple example can be found in listing 3.

```
struct Task {
  Filter trackCuts = aod::track::pt > 0.1f;

  void process(Tracks &tracks) {
    // Only tracks passing the filter will be processed.
    ...
  }
};
```

Listing 3. Simple p_{T} cut example

Finally, integration with ROOT is provided both via some helper functions to simplify reading and creation of ROOT Trees and Histograms and by providing an Arrow compatible RDataSource, which allows us to integrate our table based data model with RDataFrame [12], giving the user access to the familiar ROOT environment. Similarly, thanks to Apache Arrow ubiquity, integration with common python analysis tools like Pandas [13], plotting tools like Matplotlib [14] or machine learning packages like Tensorflow [15] is planned.

3.2 New trains infrastructure

In order to run our analysis, an upgraded the trains infrastructure is planned, modernizing the GUI front end exposed to the users and adapting it to exploit some of the facilities the new framework provides.

In particular, the ability to run tasks in parallel seamlessly is one of the most interesting aspects, together with the ability to dynamically change the train content in term of wagons.

Moreover, compared to the current implementation, not only the topology of the train will be exposed to the infrastructure, but also the configurable parameters and the input and outputs in terms of data type. This is thanks to the fact that such information is easily provided by the DPL. This will allow the train infrastructure to better introspect trains and facilitate their optimal composition and the bookkeeping of both the data and metadata resulting from running a train.

4 Conclusions and future work

While still under heavy development, ALICE O^2 Analysis Framework for Run 3 has demonstrated ability to port non-trivial analyses from the Run 2 environment, fully reproducing earlier results.

A complete prototype of a heavy-flavour analysis that uses the O^2 framework is indeed already in place. The current implementation uses all the utilities described above to perform track selection, secondary vertex reconstruction and heavy-flavour candidate selection.

As an example, in Fig. 1, the invariant mass spectrum of D^0 candidates from Monte Carlo simulations obtained with the legacy Run 1 framework and with the O2 framework is presented.

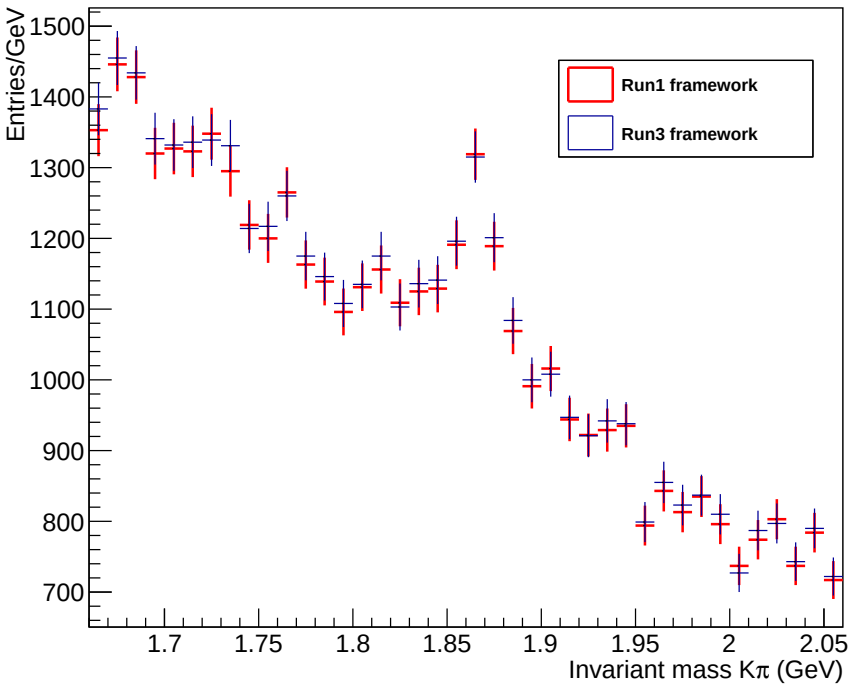

Figure 1. Invariant mass distribution of D^0 candidates from a Pb–Pb Monte Carlo simulation performed using the legacy Run 1 software and the new O2 framework.

The development will continue in the directions described in these proceedings and the framework will shortly be exposed to a larger audience of physicists. The goal is to perform a veritable data challenge by the summer of 2020.

Our future efforts will mostly consist in extending the framework to simplify porting more complex analyses, profiling and performance optimization. In particular, the aim is to demonstrate how the vendor-agnostic Apache Arrow platform allow us to seamlessly integrate a number of open source tools in a coherent, parallel and distributed environment.

References

[1] The ALICE Collaboration, Journal of Instrumentation **3**, S08002 (2008)

[2] P. Buncic, M. Krzewicki, P. Vande Vyvre, *Technical Design Report for the Upgrade of the Online-Offline Computing System* (2015)

[3] G. Eulisse, P. Konopka, M. Krzewicki, M. Richter, D. Rohr, S. Wenzel, EPJ Web Conf. **214**, 05010 (2019)

[4] R. Brun, F. Rademakers, Nucl. Instrum. Meth. **A389**, 81 (1997)

[5] S. Bagnasco, L. Betev, P. Buncic, F. Carminati, C. Cirstoiu, C. Grigoras, A. Hayrapetyan, A. Harutyunyan, A.J. Peters, P. Saiz, Journal of Physics: Conference Series **119**, 062012 (2008)

[6] L. Musa, Tech. Rep. CERN-LHCC-2012-013. LHCC-P-005, CERN, Geneva (2012), https://cds.cern.ch/record/1475244

[7] C. Jones et al., *FWCore/SOA* (2019), https://github.com/cms-sw/cmssw/commits/master/FWCore/SOA

[8] M. Schiller et al., *Soacontainer* (2019), https://gitlab.cern.ch/LHCbOpt/SOAContainer

[9] The Apache Arrow team, *From the apache arrow wiki: Physical memory layout* (2016), https://github.com/apache/arrow/blob/master/format/Layout.md

[10] Dremio, *Introducing the gandiva initiative for apache arrow* (2018), https://www.dremio.com/announcing-gandiva-initiative-for-apache-arrow/

[11] LLVM Foundation, *The llvm compiler infrastructure* (2019), https://llvm.org

[12] E. Guiraud, A. Naumann, D. Piparo, *TDataFrame: functional chains for ROOT data analyses* (2017), https://doi.org/10.5281/zenodo.260230

[13] W. McKinney, *Data structures for statistical computing in python*, in *Proceedings of the 9th Python in Science Conference* (Austin, TX, 2010), Vol. 445, pp. 51–56

[14] J.D. Hunter, Computing in Science & Engineering **9**, 90 (2007)

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin et al., *TensorFlow: Large-scale machine learning on heterogeneous systems* (2015), software available from tensorflow.org, https://www.tensorflow.org/