Apache Arrow & TDataFrame

Giulio Eulisse (CERN)

22 Mar 2018

Apache Arrow: the project

"Apache Arrow is a cross-language development platform for in-memory data"

Well established. *Top-Level Apache project backed by key developers of a number of opensource projects: Calcite, Cassandra,* ***Drill****, Hadoop, HBase, Ibis, Impala, Kudu, Pandas, Parquet, Phoenix, Spark, and Storm.*

Very active. *119 contributors, https://github.com/apache/arrow*

Why is ALICE looking into it?

- *Zero-Copy buffer adoption (well suited for our shared memory backed message passing).*
- *Interoperability with other tools (e.g. Pandas, Spark, Tensorflow, ..)*
- *ROOT interoperability is of course keystone.*

Apache Arrow: technical details

In memory column oriented storage. *Full description https://arrow.apache.org/docs/memory_layout.html. Data is organized in* `Table`*s. Tables are made of* `Column`*s.* `Column`*s are* `(<metadata>, Array)`*. An* `Array` *is backed by one or multiple* `Buffer`*s.*

Nullable fields. *An extra bitmap can optionally be provided to tell if a given slot in a column is occupied.*

Nested types. *Usual basic types (*`int`*,* `float`*, ..). It's also possible (via the usual record shredding presented in Google's Dremel paper) to support nested types. E.g. a* `String` *is a* `List<Char>`*.*

No polymorphism. *The type in an array can be nested, but there is no polymorphisms available (can be faked via nullable fields).*

Integrating Arrow and TDataFrame: TArrowDS

TArrowDS. *Arrow is a perfect match as TDataFrame backend, so I wrote* `TArrowDS` *mimicking* `TCsvDS`*.*

Initial development. *Quickly done thanks to the hints from Danilo and Enrico. https://github.com/root-project/root/pull/1712. Roughly 3 full days of development.*

Would be nice extensions. *A lazy version of* `TArrowDS` *is probably a good idea and I will probably write one at some point.*

TDataSource: Feedback

Overall, easy to use API. *My comments are really on how to improve things further, not compliants. I am already really pleased with the current API.*

Bulk API. *Current API is entry by entry, however very often you can guarantee at least partial contiguous entries so it would be nice to have a bulk API.*

```
auto [ptr, minEntry, maxEntry, size, stride] = \
  SetEntryBulk(slot, attemptedMin, attamptedMax);
```

Non rectangular sources. *Right now all the columns need to have the same set of entries and it's a construction time imposition. IMHO, it would be nice to to have such a check imposed when performing an action on given columns, not at data source construction time.*

TDataFrame: Feedback

TDataSink. *For my usecase, I need to pass results downstream to a different device via a memory mapped region. I can of course have a* `Foreach()` *to copy them back to a* `arrow::Array`*. It would be nice to be able to write directly to an* `arrow::Array` *or at least to get pointer I can adopt in a* `arrow::Buffer`*.*

Support for dynamic partitioning. *Given a collection of N objects (say, tracks), spanning M events, I want to be able to say:*

- *Group tracks by event.*
- *Apply reduction function on each group of tracks.*
- *Store results in a separate column.*

Support for emitting more than one result. *Given a collection of N objects I want to be able to emit more than one result which will end up as consecutive entries in a separate column.*

Support for combinations. *Given a collection of N objects of type T and M objects of type V I want to be invoked for all the* `NxM` *combinations and emit* `f(n,m)`

Support for joins. *Given a collections of associations between elements of a column N and elements of a column M, for each one of the associations calculate* `f(n,m)`*.*