RForest: Evolution of ROOT TTree I/O

Jakob Blomer, CERN

ROOT Users' Workshop 2018, Sarajevo

Agenda

Motivation

File Format Exploration

I/O Subsystem Decomposition

Status and Outlook

Motivation

The TTree

TTree's column-wise format is performance-engineered for our very problem!

- Only few other column-wise formats
  - Apache Parquet (Google Dremel)
    optimized for deep, sparse collections: our data is not sparse
  - Apache Arrow: transient, in-memory format
- Performance and file size compared to many other file formats
- ROOT's unique feature: seamless C++ integration
  Users do not need to write or generate schema mapping

Serialization of Nested Collections

```
struct Track {
  int fVertexId;
};

struct Particle {
  float fPt;
  std::vector<Track> fTracks;
};

struct Event {
  int fType;
  std::vector<Particle> fParticles;
};
```

We want to ensure that ROOT I/O continues to yield the most efficient analysis I/O.

TTree's column-wise format is performance-engineered for our very problem!

- Only few other column-wise formats
  - Apache Parquet (Google Dremel) optimized for deep, sparse collections: our data is not sparse
  - Apache Arrow: transient, in-memory format
- Performance and file size compared to many other file formats
- ROOT's unique feature: seamless C++ integration
  Users do not need to write or generate schema mapping

Serialization of Nested Collections

```
struct Track {
  int fVertexId;
};

struct Particle {
  float fPt;
  std::vector<Track> fTracks;
};

struct Event {
  int fType;
  std::vector<Particle> fParticles;
};
```

We want to ensure that ROOT I/O continues to yield the most efficient analysis I/O.

RForest: Investigating the Future Path of TTree

1. Speed
   - Improve mapping to vectorized and parallel hardware
   - For types known at compile / JIT time: generate optimized code
   - Optimized for simple types (`float`, `int`, and vectors of them)
   - Optimized integration with RDataFrame
2. Robust interfaces
   - Compile-time safety by default
   - Decomposition into layers:
     Logical layer, primitives layer, storage layer
   - Separation of data model and live data

Ansatz

The `RForest...` classes provide a small subset of the TTree and are used for code experiments, for instance with LHCb Run 1 OpenData examples and CMS NanoADOs. ▸ github

Interface Sketch

Writing

```
auto eventModel = std::make_unique<RTreeModel>();
auto particleModel = std::make_shared<RTreeModel>();
auto pt = particleModel->Branch<float>("pt");
auto particles = eventModel->BranchCollection(
  "particles", particleModel);

// With cling:
//auto event = eventModel->Branch<Event>();

RColumnOptions opt;
RTree tree(eventModel, RColumnSink::MakeSink(opt));

for (/* events */) {
  for (/* paricles */) {
    *pt = ...;
    partilces->Fill()
  }
  tree.Fill();
}
```

Reading

```
RColumnOptions opt;
RTree tree(RColumnSource::MakeSource(opt));
auto view_particles =
  tree.GetViewCollection("particle");
auto view_pt = view_particles.GetView<float>("pt");
for (auto e : tree.GetEntryRange()) {
  for (auto p : view_particles.GetRange(e)) {
    cout << view_pt(p) << endl;
  }
}
```

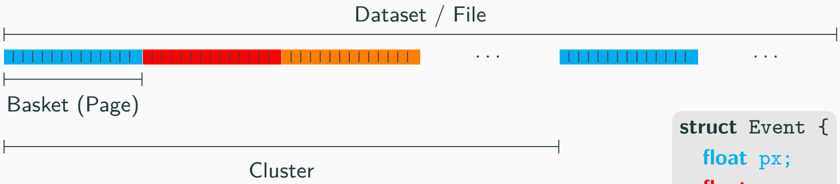
RDataFrame

```
RColumnOptions opt;
opt.pathName = ""; // ...
auto rdf = ROOT::MakeForestDataFrame(opt);
```

Not meant for release but for experimentation under real conditions.

File Format Exploration

Breakdown of the Columnar Format




```
struct Event {
  float px;
  float py;
  float pz;
};
```

Basket / Page

- Unit of writing
- Unit of (de-)compression, except for `zstd`
- Unit of vectorization and bulk I/O
- Unit of reading when small reads are cheap

Cluster

- Block of (complete) events
- Unit of parallelization (read and write)
- Unit of reading when small reads are expensive

On object stores, we can map pages or clusters to objects (to be investigated)

Format Changes in RForest Compared to TTree

1. Little-endian, which matches most contemporary architectures
2. Separate baskets/pages with values from baskets/pages with indexes pages for (nested) collections

```
struct Particle {
  float fEnergy;  // plot only fEnergy...
  float fCharge;
};
struct Jet {
  std::vector<Particle> fParticles;
}
struct Event {
  std::vector<Jet> fJets;
};
```

Potential gains of the refined layout

- Natural access to bulk I/O

  First experiments indicate an improvement of the order of factor 5 in de-serialization
- Reading can return a reference to the memory buffer, avoiding value copies

  First experiments indicate an improvement of the order of factor 2 in de-serialization
- Branches of deeply nested collections benefit from columnar access

  Significant speedup but for a small subset of analyses – no additional cost introduced

Note: the reading speed is affected by both deserialization and decompression

Other Areas of Interest

- Memory management of I/O buffers: can we stay within a fixed memory budget
- Asynchronous interfaces and scheduling of I/O transfers
- Compression algorithms:
  for instance, is it worthwhile applying different compression algorithms to different branches
- Clearer separation of I/O operations (transfer, decompression etc), reduction of their synchronization points

I/O Subsystem Decomposition

I/O Subsystem Decomposition

Logical layer
cling-assisted mapping of C++ types onto columns
e.g. `std::vector<float>` ↦ index column and a value column
or BLOB ↦ size column and `unsigned char` column

Primitives layer
"Columns" containing elements of fundamental types (float, int, ...)
grouped into (compressed) pages

Storage layer
e.g. TFile, raw file, object store

Static	Live
RTreeModel	RTree
RColumnModel	RColumn

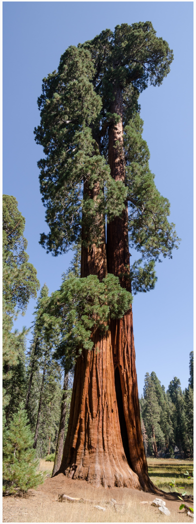
Separates the schema from the data; e.g., signal tree and background tree from same schema

- Allows for measuring performance of individual layers
- Allows us to experiment with different storage backends
- Primitives layer decoupled from C++ type system allows for lightweight 3rd party readers

Status and Outlook

Summary & Outlook

- "RForest" is exploring the evolution of the TTree I/O
- Aims at matching future analysis demands and storage systems
- Optimize for simple event models à la NanoAOD
- "RForest" provides a clean slate test environment for realistic experiments
    - Allows for investigating different parts of the I/O individually
    - Allows investigating several approach to select the ones that find the way into ROOT



Backup Slides

File Format Checklist

Functional core requirements:

- ✓ Clusterized, columnar physical layout
- ✓ Support nested collections
- ✓ Machine-independent (de-)serialization
- ✓ Recovery from canceled data set writes
- ✓ Support for different compression algorithms
- ✓ Tunable for different storage classes (SSD, HDD, Network)
- ✓ Schema evolution