A novel dynamic event data model using the DRILLBIT Column Store

TNG TECHNOLOGY CONSULTING

Johannes Ebke

Peter Waller

ScraperWiki

Find us on github!

drillb.it

If there was a method to make large physics collaborations much more productive, how could it look like?

To increase the productivity of a system, you have to **reduce the work** that it has to do.

Enable people to work **independently** on problems, but also make sure it is possible to **combine** their work again.

Make **reusing** data easier.

Reduce communication and coordination overhead.

Remove unnecessary complexity.

Automate consistency checks.

Concept: Dynamic Columnar Event Data Model

Technical Inspiration

Dremel: Interactive Analysis of Web-Scale Datasets
http://www.vldb.org/pvldb/vldb2010/pvldb_vol3/R29.pdf

- Describes a **columnar data layout** and multilevel excution trees used for fast querying of web-scale datasets at Google
- Impressive performance numbers: Aggregation queries on **24 × 10^9** nested records with 530 fields each in a datacenter with 2900 nodes run in **~2s**

Structural Inspiration

Git: Distributed Version Control

- Uses a **content-addressable store** to **guarantee** that the given version (hash) corresponds to the data in your workspace
- Commits and changes can be made in a **distributed** way, with each committer working independently, and then merged
- Tags from different people can be put into one repository - even if they have identical names - using **namespaces**

Model

- Similar to more heavyweight experimental software, data can be modelled using **named objects** and variables
- Algorithms can be specified either using **C++** as usual or by directly specifying simple operations in a **simple DSL** similar to TTree::Draw() syntax
- Event, object and column unique identifiers and hashes allow **safe cross-column links** to be implemented
- Which algorithms or calculations are applied is saved in the metadata of created columns, possibly using a hash from a central git repository. This enables **full reproducibility** of calculations and a better understanding of the final results.

Dynamic

Event fields and objects do **not** have to be universally agreed upon. Physicists can **pick and choose** the most appropriate. **Namespaces** provide "official" versions of objects.

Columnar

- Each data field of an event or of a sub-object - a "column" - can be **stored separately**
- Columns can be **packed into zip files**, while still allowing direct access using seeks or HTTP vector reads (supported e.g. by dCache)
- Already widely used in HEP as TTree branches

Data

The data itself is stored in **plain binary files** or packed in **zip files**. No new technologies for file storage or management are proposed. **Metadata** is always stored alongside the files, and **no central server** or database is required for analysis.

Event

An event or object is identified by its **index** inside a data block, which is identified by a **unique ID** or hash. In addition, each column has a **version hash** to identify it across data blocks. Columns belonging to objects generated by the same algorithm also have a unique object identifier. This ensures **data always matches up**.

Example EDM Configuration

```
muon := perf/muon@r2321
muon.isolation := HxMuonIsolationTool@v00-02-03(muon)
muon.momentum := git://my_tool@r42(muon, event_info)

   ./drillbit-get-from-grid-or-calculate -c config
```

Data Layout

Pure array of data, e.g. as float32 → Column Data + Meta Data

- Default Name
- Datatype
- Block UUID
- Object UUID(s)
- Depth/Level information: ~0-4 bit/datum

Current Situation

transient data

persisted data

bugfix

correction

few developers

$ little money

My Dataset

```
TTree*t=new TTree("foo");
t->SetBranchAddress();
f = TFile("test.root")
t->Branch("px", "I", "px/I");
t->Branch("py", "I", "py/I");
t->Branch("pz", "I", "pz/I");
```

we cannot agree on "the data"

complex machinery

based on code ...not based on data

New Concept

lots of data

d at a & d at a

BUGFIX

CORRECTED

COMMENTUM

My Dataset

- Split large tuples into columns,
- One or more columns per variable
- Efficiently combine columns into tuples on request
- Add new columns instead of reprocessing or running manual fixes
- Reduce the immense pressure on developers to get (almost) everything right in the first go in the (re)processing

Code Status Summary

- columns implemented for all basic data types in **2kLOC C++**
- gain of **-20% in size** w.r.t ROOT (same zlib)
- **TTree-compatible** processing interface, with speed on par with native ROOT
- Processing of single columns extremely **fast**.