Rethinking the Data Model: The Drillbit
Proof-of-Concept Library

Johannes Ebke\textsuperscript{1} and Peter Waller\textsuperscript{2}

\textsuperscript{1}TNG Technology Consulting, Betastr. 13a, 85774 Unterföhring, Germany
\textsuperscript{2}ScraperWiki, Liverpool Science Park, 146 Brownlow Hill, Liverpool L3 5RF, UK

E-mail: johannes@ebke.org, p@pwaller.net

Abstract.

The focus of many software architectures of the LHC experiments is to deliver a well-designed Event Data Model (EDM). Changes and additions to the stored data are often expensive, requiring large amounts of CPU time, disk storage and man-power. In addition, differing needs between groups of physicists lead to a tendency for common data formats to grow in terms of contained information whilst still not managing to service all needs. We introduce a new way of thinking about the data model based on the Dremel column store architecture published by Google. We present an EDM concept based on Dremel, which has the potential to significantly reduce the storage requirement for these common formats, decrease the time needed for independent physicists to compare their results and improve the speed at which data reprocessings can feasibly take place. The Dremel low-level encoding is implemented in a proof-of-concept C++ library called Drillbit, and it is shown that using a different encoding of the current data could save as much as 20\% of disk space on average across a wide number of real-world derived data sets.

1. Introduction

The LHC has been collecting a large amount of data over the last run, and much more is coming. The quality of the physics that can be done is impacted by the limited computing resources due to the constraints on the amount of data that can be stored and the processes through which it is analysed.

So far in 2013, the general purpose LHC experiments ATLAS and CMS have recorded of order $10^9$ proton collisions each [1], and have simulated similar numbers of collision events. For the two general purpose experiments the current storage requirements for these events are each of order 100 PB [2].

The data recorded by the detector is referred to as “RAW”, where all of the information that was read out is retained. The data are reduced and transformed into a form which is more useful for analysts (“Analysis Object Data”, AOD) representing the physics objects after reconstruction. The physics event is reconstructed using knowledge of the full detector model through many complex algorithms using a “Blackboard” model of computation [3]. Reconstruction is a fundamentally expensive process which operates at $O(10$ sec/event$)$. The events and objects belonging to a given event are represented as C++ objects. The event data are ultimately stored using a software framework called ROOT [4], which allows direct
serialization, in row or column order, of arbitrary C++ objects for which reflection dictionaries have been generated.

The first results of the reconstruction are usually stored in files, 1-20 GB in size, containing 10,000-100,000 events. Within a large physics experiment there are many physics activities which each require different types of analysis-level data. At a given moment an analyst may only require hundreds of variables, using one set of analysis object definitions, whereas the full set of event information may be \( \approx 10,000 \) variables\(^1\). Therefore many partially orthogonal reduced datasets are derived from the analysis object data by independent groups and people in the collaborations. Some of these may be derived through a centralised mechanism, or they may be generated by people for their own use, potentially from other derived datasets. These datasets are frequently referred to as “ntuples” and, for ATLAS, collectively take up more space than the analysis object data at time of writing\(^2\).

Three popular methods to reduce a dataset are:

- Remove events (skimming)
- Remove variables from the event (slimming)
- Remove objects from the event (thinning)

A large part of the analysis activities necessary for physics results centre around calibrating the detector, calculating correction factors or additional analysis variables and providing estimates of uncertainties. These activities are highly decentralised and occur in parallel. Corrections due to detector calibrations are usually already incorporated in the first full reconstruction. Many other correction factors, variables and uncertainty estimates frequently only become available shortly before the deadlines to coming conferences since they are calculated using the same data which is used in the physics analyses. These factors are often computed on a per-event and/or per-object level using ad-hoc software written in C++. This software typically takes a set of input variables which are available in the monolithic dataset. If a reduced dataset is created by removing variables, there is a significant danger that it may lack a required input for future corrections. The ad-hoc nature of correction packages and the requirement to manually provide exactly the right input make it very easy to introduce unwanted differences into analysis code, which has to be meticulously cross-checked in e.g. cut-flow comparisons\(^2\).

In addition to corrections on the analysis level, improvements and bug-fixes are continuously implemented in the event reconstruction software. These fixes are usually applied when reading in the analysis object data, and are exported to the derived datasets. When the CPU time necessary to run the fixing process increases, either a full re-reconstruction campaign or – as a short-term solution – a reprocessing of the most popular derived datasets may be performed.

To summarise: “monolithic” datasets, where at each stage information about an event is contained in exactly one file\(^3\), are the basis of current computing models. The varying requirements of analysts lead to several types of derived datasets, which may represent differing physics object definitions. On the other hand, calculating correction factors and uncertainty estimates requires, in all derived datasets, a large set of additional variables which are not known ahead of time and which don’t necessarily overlap between analyses.

---

\(^1\) We take here the union of variables of interest over analysis groups, as evidenced by the number of variables in common analysis ntuples.

\(^2\) A procedure of comparing numbers of events surviving at each stage of an analysis. Also classified as cruel and unusual punishment for PhD students.

\(^3\) We are aware that there are methods for back-navigation between stages. They are however rarely seen in practice owing to the additional effort required to use them.
repeated message event {
    int32 run_number;
    float32 total_energy;
    repeated message electrons {
        double pt, eta, phi;
        repeated double hits_timing;
    }
}

Figure 1. Document structure of a simple event model description, and record- and column-oriented storage formats. The black boxes represent information which occurs at most once per event, the blue boxes represent repeated objects, for example electrons. The red boxes represent information on the repeated objects whose size can vary between events.

2. The Dremel storage format

It turns out that problems of similar scale have been faced by Google in processing large-scale data. In [5], engineers describe a system called “Dremel”, with good performance numbers: an aggregation query (i.e. a histogram of a variable) over $24 \times 10^9$ documents (events) runs in 3 seconds on an already loaded cluster with 3,000 cores. The documents have 500 variables, which may be switched out for new versions of the variables at a small cost in computing resources.

Similar amounts of computing power are routinely available to HEP analysts. All numbers – except the processing time – are comparable to the problems encountered in HEP, which is a hint that the turn around time of particle physics analyses could be accelerated.

Unfortunately the paper was not accompanied by any code implementation, and no open source implementation was available at the time. For this reason, we produced a proof-of-concept library in C++, available at drillb.it [6], which can be used to convert ROOT TTrees containing (possibly nested) `std::vector`s into files containing Dremel-encoded data and back. All results in the following are based on this implementation.

The key to the Dremel storage format is in its novel encoding of nested documents. As a first step, documents or classes structured as shown in the left hand side of Figure 1 are represented on disk as columns, as shown on the right hand side. In ROOT, this step is called splitting the event into branches. For columns of simple variables as `event.run_number` or `event.total_energy` in this example, the data can then be written to disk as an array, since exactly one value is written for each event. For variables as `event.electrons.pt` which correspond to a C++ `std::vector`, the number of values may vary for each event. The usual way to encode this is to store the length of the vector, followed by the values within.

In the Dremel encoding however, two output streams are used, one solely for the data values, and one containing metadata about these. In the case of `event.electrons.pt` and no data reduction, one bit would be necessary for each written data value: zero if the value is the first one in a new event, one otherwise. How the encoding schemes differ is shown in Figure 2.

This scheme has several advantages over storing lengths and values intermixed. Firstly and most importantly in the short term, the compressibility of the encoded byte-stream with schemes such as ZLIB is markedly improved, especially if the vectors to be encoded have $O(10)$ elements or fewer. In a comparison with ROOT using the lowest ZLIB compression level for both methods,

4 In the meantime, at least two Java-based open source projects also use the Dremel encoding, Apache Drill [7] and Parquet [8].

5 An additional saving is possible where the same variables are kept for the same objects: the metadata streams are the same. This optimization was not yet used for the results in this paper.
Event 1: 3 electrons, pt: 42.11, 2.21, 16.24
Event 2: 2 electrons, pt: 44.12, 12.31

encoded as length followed by values:
03 00 00 00 a4 70 28 42 a4 70 0d 40 85 eb 81 41 02 00 00 e1 7a 30 42 c3 f5 44 41

Dremel encoding (metadata followed by data):
00 01 01 00 01
a4 70 28 42 a4 70 0d 40 85 eb 81 41 e1 7a 30 42 c3 f5 44 41

Dremel encoding of electrons with a pt > 15:
10 01 11 10 01
a4 70 28 42 85 eb 81 41 e1 7a 30 42

Figure 2. Simple encoding examples for the event.electrons.pt column from the example in Figure 1. In both encodings shown, the hexadecimal numbers represent uncompressed bytes in memory. This makes it obvious how the Dremel encoding scheme separates the highly compressible metadata (lengths) from the much less compressible values. Note that the metadata here could actually be represented in less than one byte if a more efficient coding scheme is used. In the last example, a zero in the first half of the metadata byte denotes a missing value.

an example data with vectors of floats from 0-1000.0 and of length 0-5 (both equally distributed), the Dremel-encoded data is 25% smaller than the same data encoded using the current ROOT encoding. The ROOT basket size was varied from the default to greater than the size of the data, the ROOT file being ≈ 2% smaller in the case of a large basket size.

The reason for this discrepancy is illustrated in Figure 2. By looking at the bytes it is apparent that the structure of the length is different from that of the data. The data represents the output of experimental measurements and is frequently close to incompressible, whereas the length parameter is a small integer, which is in principle relatively compressible. Mixing the two types of information prevents an efficient representation and compression of the latter. As a consequence of the efficient representation of the length in the Dremel scheme it turns out that some uncompressed Drillbit columns are smaller in size than the same information using ROOT with compression.

To evaluate the potential gain for experiments, we converted all official ATLAS derived analysis ntuple data for a typical data-taking run in 2012 from ROOT-encoded std::vectors into the Dremel encoding. On average, we observed a reduction in size of 20%, confirming that this issue is highly relevant for experiments.

Another advantage of the Dremel encoding is the possibility to introduce the concept of “missing values” and “missing events”, reducing the volume of the data while keeping the positional information of individual events. An example is given in the last part of Figure 2, where the same example column is encoded again while stripping out (“thinning”) values which are smaller than 15. Missing values are left out of the data stream, but denoted with a zero in the first half of the byte in the metadata stream. The same concept applies to missing events, for datasets which are “skimmed”, so that the event-by-event alignment between skimmed and un-skimmed datasets can be preserved.

The Dremel encoding efficiently preserves the full structure of the event, even if individual data are not available for a specific object within the event. In other words, one knows how to align two datasets even if different fields or objects within the event are not populated. For example, this would enable the generation of a complete column for a corrected variable which

---

6 The example code can be found in our github repository [6] in src/apps/make_test_dit.cc
would be \( \mathcal{O}(10 \text{ GB}) \) for all data. Given any column from a skim in the Dremel format, one can then generate the skim of this new corrected variable.

The Dremel paper describes an algorithm to efficiently reconstruct full document representations from a given set of columns. This algorithm can be applied to construct arbitrary in-memory representations, e.g. C++ objects using ROOT introspection, C structs or also Google Protobuf [9] messages.

To conclude, the Dremel representation of nested data provides real near-term benefits in terms of data size. The data size of physics data can possibly be reduced by \( \approx 20\% \). The concept of “missing values” allows reduced datasets or columns to be in principle compatible with the full dataset and also each other, enabling new work-flows which will be discussed in the following section.

3. A dynamic event data model
3.1. Distributing columns across files
Using the Dremel encoding makes one work flow more attractive: splitting the data columns into separate groups on disk. This feature is already available in ROOT as “friend trees”, but to our knowledge it has not yet been exploited at a large scale. From personal experience, we believe the missing built-in safeguards against friend tree mismatches – both on the event level and on the file level – are one of the reasons for this. In addition, reduced datasets, be they slimmed, skimmed or thinned, can not be matched up against each other or against the full dataset, unless a common unique key and index lookups – with their associated performance and storage cost\(^7\) – are used.

One possible objection to using such a scheme on a large scale is the question of how to locate the files required for a given analysis, since more than one file may be required simultaneously for a given event. This may seem problematic since the product of the number of columns required for physics (\( \mathcal{O}(10,000) \)) and the number of files (\( \mathcal{O}(10,000) \)) is \( \mathcal{O}(100,000,000) \). The probability that some number of files are unavailable due to a network or disk failure is high. Missing data can be represented in a natural way in the Dremel architecture, so the remaining problem is how to efficiently determine which files should be processed in the same job.

To provide some intuition on this issue, we wrote a proof-of-concept dispatcher\(^8\). For each set of events (e.g. a physics run), the dispatcher finds the set of files that contain all columns needed for processing, even if files contain different numbers of columns or events. For example, consider the case of 10,000 columns which are split across 10 files for each range of events\(^9\). When a single column is reprocessed this may only be a kilobyte of data per range of events. If each range is stored in a separate file, this would represent a significant storage overhead. It therefore makes sense to store this column for many event ranges in one file. The proof-of-concept dispatcher demonstrates that the problem can be solved rapidly (\( \mathcal{O}(10 \text{ s}) \) for 10,000 columns and 1,000 event ranges) whilst also determining the set of event ranges for which data is missing in any of the columns.

3.2. Semantic mapping of variables
The first step in any final stage analysis is to map the available variables – measured quantities – onto the physics objects under study. As an example, one may choose the electrons \( p_T \) as measured by the tracking system, or by the calorimeter, or by a combination of the two. The

---

\(^7\) Consider the case of a single additional variable where a key composed of \( \text{(run number, event number, object index)} \) is required per datum in order to match these objects across different selections. This key may then be even larger than the data. In the Dremel encoding it can be represented much more efficiently, see the end of Section 2.

\(^8\) Available on drillb.it [6] in proof_of_concept/dispatch_simulator

\(^9\) For example, 10 classes of physics object with 1,000 variables each.
choices multiply for jets, where usually a multitude of reconstruction algorithms are available. Over time, the choices may also evolve depending on changing requirements of the analysis and available calibration data. It therefore is convenient to enable the physicist to make these choices in an explicit manner.

One possible form for this wiring would be an initialization file, containing mappings like:

\begin{verbatim}
event.electron := event.calibrated_electron
event.electron.pt := event.calibrated_electron.pt_tracker
\end{verbatim}

These mappings could then be used in the initialization phase to locate the required columns on the right-hand sides, mapping them to files and connecting them semantically to the variables and objects on the left-hand side. This would enable modifying the input data that a common tool uses without having to modify the source code and recompile. Interoperability between analyses could be achieved with less effort.

Taking one step further towards reproducibility and ease of sharing input data, one could put versioning information into such a file:

\begin{verbatim}
event.electron := event.calibrated_electron@chep2013
event.electron.pt := event.calibrated_electron.pt_tracker@calibration42
\end{verbatim}

Taken together, if an analysis tool (containing an initialization file) again produces one or more variables, references to its source code repository could replace references to its results in the right-hand side of a specification:

\begin{verbatim}
event.electron := git://experiment.ch/tools/electron/calibrator@calibration42
\end{verbatim}

This would allow the initialization routine to either download the required data directly or decide to re-run the tool if the input needed by the tool is available. Reduced datasets could also be used automatically, if information about the reduction process (cuts) is stored alongside the reduced datasets.

At this point, such a system would be useful for physicists running final-stage analyses, since with the right tools they could specify exactly which data they want (and share these specifications), and run a simple command that retrieves the data locally into a format of their choice, e.g. a ROOT tree\cite{10}. In addition to being useful for analysts, this approach would also allow for reprocessings to produce updated versions of a chosen subset of columns. Since currently a reprocessing doubles the disk space required by the data, this would lead to significant savings in storage, and enable reprocessings – updates of column versions – much more frequently. As an example, a few numbers for each of $10^9$ events would only occupy $O(10 \text{ GB})$ of space.

If they were incorporated into the computing model and software framework of an experiment, the flexible and loosely coupled manner of column specification would also enable a decentralization of the reconstruction process: since creating a new column with few variables is computationally cheap, it can be done as part of the testing process for new tools. Eventually, updating to a new analysis software version would be replaced by tagging a new, already prepared version of the data.

3.3. Event Model Summary

Although many more enticing long-term possibilities of this architecture present themselves, we will conclude with summarizing the short-term steps that seem possible today. A useful initial project would be a service that runs on columnar datasets to produce, on demand, reduced datasets in ROOT form. This would make it possible to do managed partial reprocessings, switching out columns in the service, with no impact on analysts. A second possible project

\footnote{If any of the variables have already been computed by anyone in such a system they can be integrated into the analysis at network speeds.}
would be to enable a configuration as described in Section 3.2, which simplifies explicitly mapping variable names in files to physics objects of interest. Finally, introducing improved, automatic safeguards for ROOT friend trees would already enable several of the benefits of the architecture described above.

4. Summary and outlook
If all goes according to the current plan, the LHC will start up again in early 2015. The LHC experiments will again be faced with the challenges of managing both large amounts of data and large numbers of physicists. Existing software and data architectures require a high level of coordination from the users; such as choosing a superset of common variables which satisfy the need of all analysts and distributing code to compute corrections. This is getting more burdensome for the collaborators as collaborations get larger. An event data model with a greater focus on allowing independent columns could reduce the coordination requirements significantly while also reducing data storage requirements.

The Dremel encoding is designed for such an architecture. In addition to significantly decreasing the data size compared to current methods, its concept of “missing values” allows using reduced and non-reduced datasets together in a single analysis. Explicitly mapping versioned columns on disk to physics objects allows analysts to share data definitions instead of manually wiring up tools in C++ code.

The Drillbit library was written as a proof-of-concept, to validate the usefulness of the Dremel encoding for HEP. As such, is has surpassed our expectations, validating several of the core concepts of a dynamic columnar event model. However, as both of the authors are no longer working in HEP, we have concentrated on presenting the concepts, rather than the implementation, and would invite people to either take up one of the projects described at the end of Section 3, or even use Drillbit as a starting point for the development of a new architecture. We believe that a modest investment in such an effort could quickly show whether this is a viable concept and potentially catalyse motion towards the next order of magnitude improvement in disk space, analysis time, and physicist productivity.

Acknowledgments
The authors would like to thank Wahijd Bhimji, Elizabeth Gallas, Paul Laycock, Axel Naumann and many other attendees of CHEP 2013 for the fruitful discussions. We are also grateful to TNG Technology Consulting for the time and funding which allowed Johannes to attend the conference. We owe a debt of gratitude to our wonderful significant others Amelie and Ona for putting up with our particular flavour of insanity.

References
[1] ATLAS Run Query Tool, retrieved on 30. October 2013 http://atlas-runquery.cern.ch/
[2] ATLAS Dashboard, DDM Accounting, retrieved on 30. October 2013
   http://dashb-atlas-ddm-acc.cern.ch/dashboard/request.py/ddmaccounting
[3] Calafiura P et al. 2003 Computing Research Repository cs.SE/0306089.
[4] Brun R and Rademakers F 1997 Nucl. Inst. & Meth. in Phys. Res. A 389 81-86.
   See also http://root.cern.ch/.
[5] Melnik S et al. 2010 Proceedings of the VLDB Endowment 3.1-2 330-339.
[6] Drillbit Library http://drillb.it
[7] Apache Drill Project https://incubator.apache.org/drill/
[8] Parquet http://parquet.io/
[9] Protocol Buffers, Googles Data Interchange Format http://code.google.com/p/protobuf
[10] Agarwal A et al. 2009 J. Phys.: Conf. Series 219 072024