LINTToRoot

M. Proffitt, E. Torro, G. Watts
(UW/Seattle)

IRIS-HEP Topical Meeting
Feb 25, 2019

(apologies for the style of this talk... I got going and couldn’t stop)
Declarative Analysis is an Old Concept

Specify what you want to do not how you want to do it

SQL, based on a CS paper from 1970, has been around since the early 70's.
Plot Missing $E_T$ for events with 2 jets with $p_T > 40$ GeV and having at least 2 2 GeV tracks within $\Delta R < 0.2$ for full Run 2 analysis.

- Data Model
- CPU Resources
- GPU Resources
- DOMA
Plot Missing $E_T$ for events with 2 jets with $p_T > 40$ GeV and having at least 2 2 GeV tracks within $\Delta R < 0.2$ for full Run 2 analysis.

The Target
- The Missing $E_T$ attribute of the event object
- Could be a tuple (Missing $E_T$, $N_{jets}$, etc)
- Implied loop over events

Selection

Note:
- *Implied* double loop
  - Then over jets
  - Then over tracks
  - Matching between levels

Imagine adding the word *Calibrated*
The Backend (Analysis System)

Data Model
- How data is structured
- How to code the “query”

DOMA
- Where to find the data
- Caching of intermediate results
- Picking up previously cached results

Scheduler
- Where to find the data
- Caching of intermediate results
- Picking up previously cached results
Production System Analysis Files

Scan data, explore with histograms, making final plots

Fitting, manipulation, limit extrapolation

Archiving, publication, Reinterpretation, etc.

Analysis Systems, analysis & declarative languages (underlying framework)

LINQToROOT

https://github.com/gordonwatts/LINQtoROOT
The electron is a first class object, specific to class of experiment.

Data model contains object definitions, data structure is part of the language, experiment agnostic.

Data model contains all information, field and experiment agnostic.
I chose Microsoft’s C# language due to built in SQL-like language, LINQ:
- Strongly typed
- LINQ is extensible to new backends by design
- Automatic tooling support
- Fully capable language with lots of Open Source libraries
- Statically typed
- Based on paper by CS theorist (who also came up with Reactive Programming)

```csharp
events
  .Select(e => e.Data.eventWeight)
  .FuturePlot("event_weights", "Sample EventWeights", 100, 0.0, 1000.0)
  .Save(hdir);
```

What we want to plot

1D Histogram Declaration

Save the plot in a file

Note: There is no explicit loop!
Where(e => e.Jets.Where(j => j.pT > 40.0 && e.Tracks.Where(t => ROOTUtils.DeltaR2(j.eta, j.phi, t.eta, t.phi) < 0.2).Count() >= 2))
.Select(e => e.MissingET)
.FuturePlot("met", "Missing ET for Events with 2 good jets", 100, 0.0, 1000.0)
.Save(hdir);
## LINQ Operations Implemented

### Sequence Operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Apply a function to each object of the sequence rendering a new sequence</td>
</tr>
<tr>
<td></td>
<td>(transformation): sequence of jet → sequence of jet.pt()</td>
</tr>
<tr>
<td>SelectMany</td>
<td>Unroll a loop: sequence of jet → sequence of tracks near each jet</td>
</tr>
<tr>
<td>Where</td>
<td>Filter: sequence of jets with pt &gt; 50 GeV</td>
</tr>
<tr>
<td>OrderBy</td>
<td>Sort by some function of each object in the sequence (ascending or descending)</td>
</tr>
<tr>
<td>GroupBy</td>
<td>Group sequence into a sequence of sequences (jets between 10 and 40 GeV, 40 and 90 GeV, etc.).</td>
</tr>
</tbody>
</table>

### Terminal Operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate</td>
<td>Accumulate, used as the basis of many other terminals: filling a histogram, calculate average, sum, etc.</td>
</tr>
<tr>
<td>All</td>
<td>True if every element of a sequence satisfies some condition</td>
</tr>
<tr>
<td>Any</td>
<td>True if any element of a sequence satisfies a condition</td>
</tr>
<tr>
<td>Count</td>
<td>Number of elements in a sequence</td>
</tr>
<tr>
<td>Empty</td>
<td>True if a sequence has no elements</td>
</tr>
<tr>
<td>First</td>
<td>First element of a sequence</td>
</tr>
<tr>
<td>Last</td>
<td>Last element of a sequence</td>
</tr>
<tr>
<td>Max, Min</td>
<td>Max or min value of some function over a sequence (you can get the object that satisfied this)</td>
</tr>
<tr>
<td>Skip/Take</td>
<td>Skip n elements or take n elements, or by a condition (take until/skip while)</td>
</tr>
</tbody>
</table>
Data Flow Through LINQToROOT

C#, UI (all text!)

Query -> Abstract Syntax Tree

Cache

AST Remapping, AST -> ROOT/C++

Dataset -> Files

ROOT/C++ -> Plots, TTrees, etc.
C#, UI
(all text!)

- Needs to be 100% text (IMHO!!)
- Easy to compose streams of data and selections
  - Too easy? 1000’s of histograms
- Provides for leaky abstraction
  - Here: including arbitrary C++ code
- Allows 1000’s of plots to be made and results manipulated with futures
  - C# is not so strong with this (monad hell)
- LINQ is based on a Data Manipulation Language foundation, so is quite complete
  - Functional
  - Histogram filling is an Aggregate step
  - Feeds the whole OO was the largest mistake in the last xx years mythos.
- Full programming language allows for manipulation of histograms (do not want multistep analysis if can help it!)
- Strongly Typed!!!! No text strings, but this does cause some pain.
The AST contains all the information for the query:
- Source dataset
- All selection cuts
- Final desired product (a histogram, TTree, etc.)
- Any data
  - A histogram that is used to re-weight events
AST is unique for a query, it contains no “opaque” code:
- This does have restrictions on what type of C# code you can use.
  - C# supports generating AST for lambda functions as part of the language standard. This is a key feature to make this work.
AST (and C#) expressive enough to do raw ATLAS xAOD.
Vision: cut/paste text representation of AST into tool and it generates the code (Data Preservation, Jupyter notebook to repro a plot, etc.). A Jupyter Notebook...
AST – could be sent over wire to Analysis System Server.
Data Flow Through LINQToROOT

- The Cache uses the AST as a key
- Product is stored (histogram in a ROOT file, csv file, etc.)
- Look up and load was less than 10 ms.
  - Bulk of time was translating ROOT histograms to a hash when they are part of the AST
  - And opening and loading a ROOT file (every histo was stored in an individual file)
  - Rerunning 1000 plots still took 4-5 minutes (still too slow).
- Cache worked across different code bases as long as AST matched
- Was localized to a single machine (unfortunately).
- Obvious place to work with DOMA folks to understand how to do this better
Remapping performs arbitrary transformations on the AST
• Take a flat ntuple and make it look like it has objects (electron pt, eta, phi are separate branches, but in C# can loop over electrons and look at pT, etc.).
• Important to let user code in way that promotes “physics” thinking and the backend to run as efficiently as possible.
• Original reason: our group was writing out flat ntuples and that drove my brain nuts.
• Many other transformations possible, but not explored.
• AST’s from different queries can be combined to make scan over data more efficient
• AST’s can be optimized (e.g. loop invariants, common calculations). This is where most of my bugs existed and bulk of unit tests (~300 or 400).
• C++ code written using the visitor pattern on the AST.
• When I ran at University of Washington or at CERN I’d get the same thing
  • Even though the files were located on different computers
• Tools to copy datasets
• Tools to submit jobs to take production skims and turn them into flat ntuples
  • Allowed me to track, with git, analysis from production system to plots
• This was an ad-hoc system!
  • Obvious place to work with DOMA folks and make something that works well
Data Flow Through LINQToROOT

- Multiple backends
  - Run on my windows machine
  - Run on a Linux partition in my windows machine
  - Run on Linux via SSH
  - Run on multiple Linux machines (e.g. a really simple PROOF).
- AST told you what leaves were going to be touched, so could pre-configure.
- For the most part, the system was very simple.
- If the C++ didn’t compile, I considered that a bug in LINQToTTree
  - The UI (or C#) should catch all errors.
  - This was to help prevent the abstraction from leaking
  - Also... C++ compiler error messages...
  - Mostly because my optimizer had bugs in it
Where to next?

- Can this be done in python?
- Can a similar language drive an xAOD, Flat TTree, and columnar pandas like analysis?
  - Confident of the first two
- Python has different idioms and abilities vs. C#
  - Can the UI be made concise and clear?
- Can an AST fully express queries?
  - Over the wire?

- Have just started a [github repo](https://github.com) to explore this.
- Prototype quality code
- Hope to have rough answers to most questions in a month or two

- Query in a simple AST
- Runs full ATLAS environment in docker
- Exports a root file and uses uproot to import it as a Pandas dataframe
Final Thoughts

- I am a firm believer in this approach
  - It is definitely not without its problems
  - For me this started as a grumpy old professor research project: make it easy!
- I think we must move towards splitting the analysis language and server
  - Common protocol for the field?
  - Even locally (everyone can run docker)
- C# as a choice
  - One of the most expressive imperative OO languages I’ve used (LINQ, pattern matching in code, garbage collected)
  - Available in Linux – but known well by a minority of physicists
  - Unfortunately, no other language has the proper set of features built into the language standard
- High level specification of analysis language
  - DOMA experts can work on back ends
  - Anyone can take advantage of their own language if they want
  - Allows independent advancement
  - AL could be quite powerful outside of HEP analyzing structured data
- Pain Points
  - 4-5 minutes to re-run 1000 histograms too slow
  - Histograms
    - Common axes as building blocks
    - Change title or axis label and binary representation changes (caching).
  - Cache was one machine only
  - What if you want to change a base line cut and look at one histo?
    - That takes 20 minutes, but all 1000 will change, and now you are looking at 5-6 hours
Backup
Sense of Scale

- Number of events (signal + background + control): ~2 billion
- GRID Data Samples: ~200
- Size of input files: 1-2 TB
- Number of leaves in processed ntuples: ~340
- Plots made per job: ~400-800
- Number of users: 1
- Number of Developers: 1

A small analysis by HL-LHC standards...
But a decent sized one for Run 1 and Run 2
Published 3 papers and 2 ATLAS CONF notes in part using this tool.