Developing New Indexing System for ATLAS Entry Data

Nikita Dulin
DOE SULI Summer 2018
Mentor: Peter van Gemmeren

Argonne NATIONAL LABORATORY

U.S. DEPARTMENT OF ENERGY

Outline

Large Hadron Collider and ATLAS

Short ROOT Introduction

- Data Storage and I/O

ATLAS data management

- New Indexing System

Means and method of incorporating new code into existing ATLAS framework

ATLAS experiment at LHC

ATLAS is one of four experiments at the LHC in Geneva

Analyzes proton-proton collisions

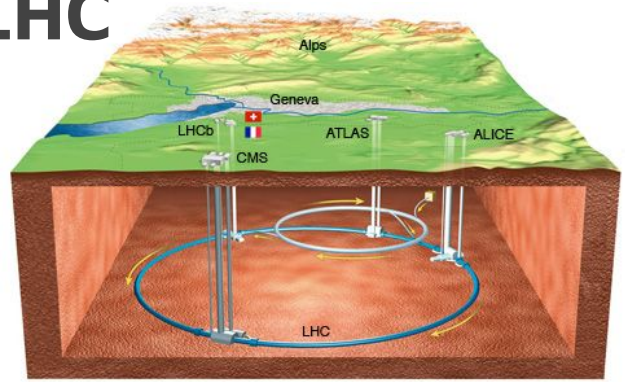
Designed for discovery of new particles

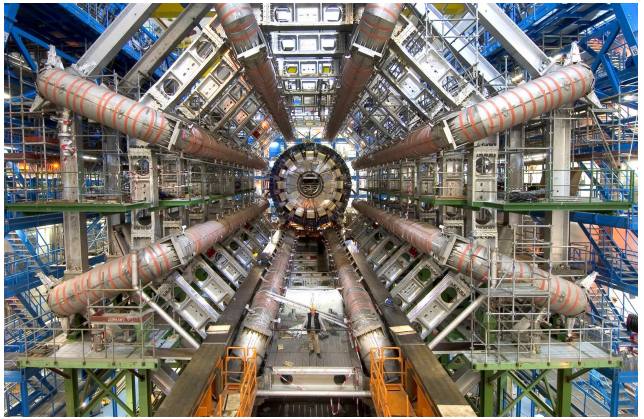
- Higgs boson

Designed to improve precision measurements

- Gauge bosons and heavy quark properties

Has produced 200-300 PB of event data



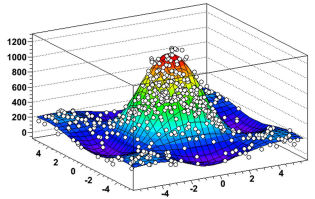


ROOT

ROOT: Data Analysis Framework

Free, open-source, object-oriented, c++ based

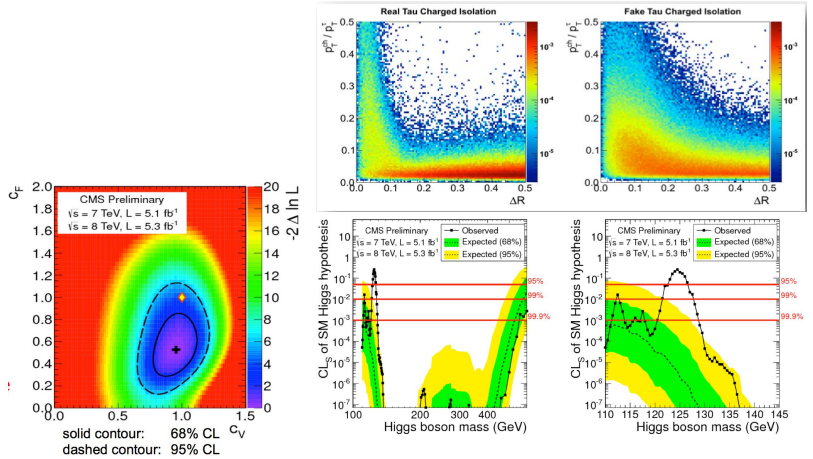
Used extensively in high energy and particle physics for 20+ years



Can efficiently handle large amounts of data

Paramount for high energy physics

Histograms, 3D Modeling, simulations

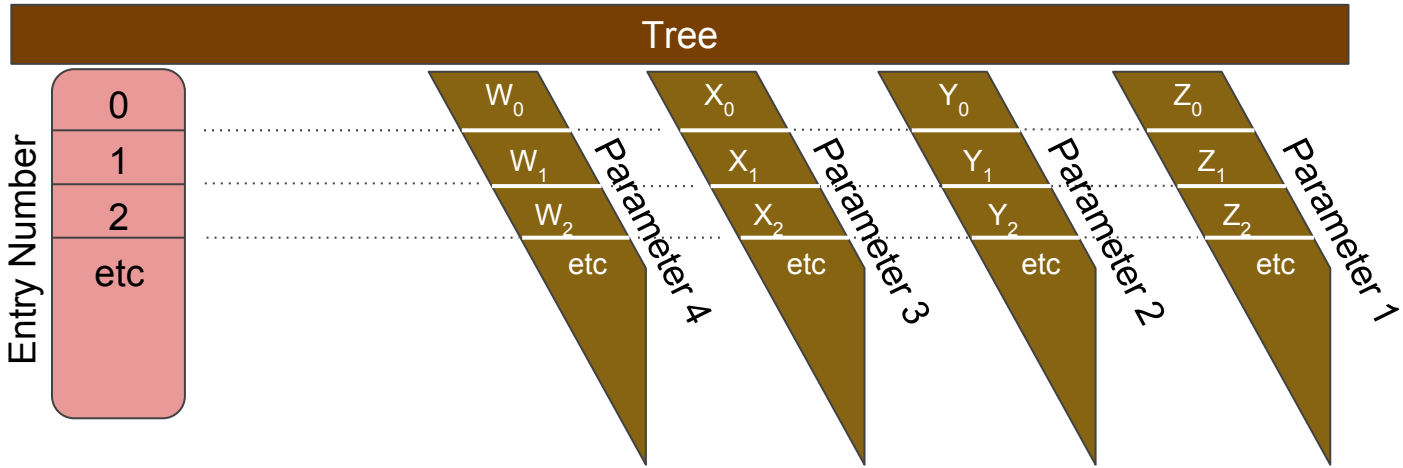


ROOT Data Storage and I/O

Data stored column-wise in TTrees and TBranches

Different branch for each parameter - different values for each entry

Basically n-tuples, each entry number referring to a single n-tuple



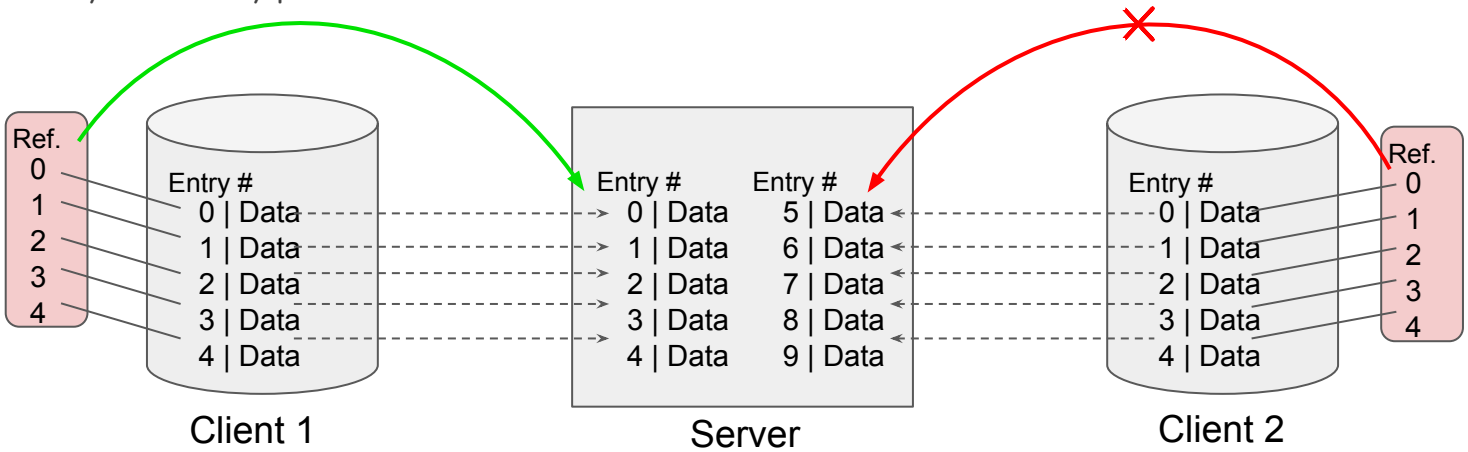
ATLAS Data Management and ROOT

Entry number is sequential from 0

ATLAS only uses entry number for referencing

In-memory merging from multiple clients resets entry numbers

Ability to identify particular events is broken

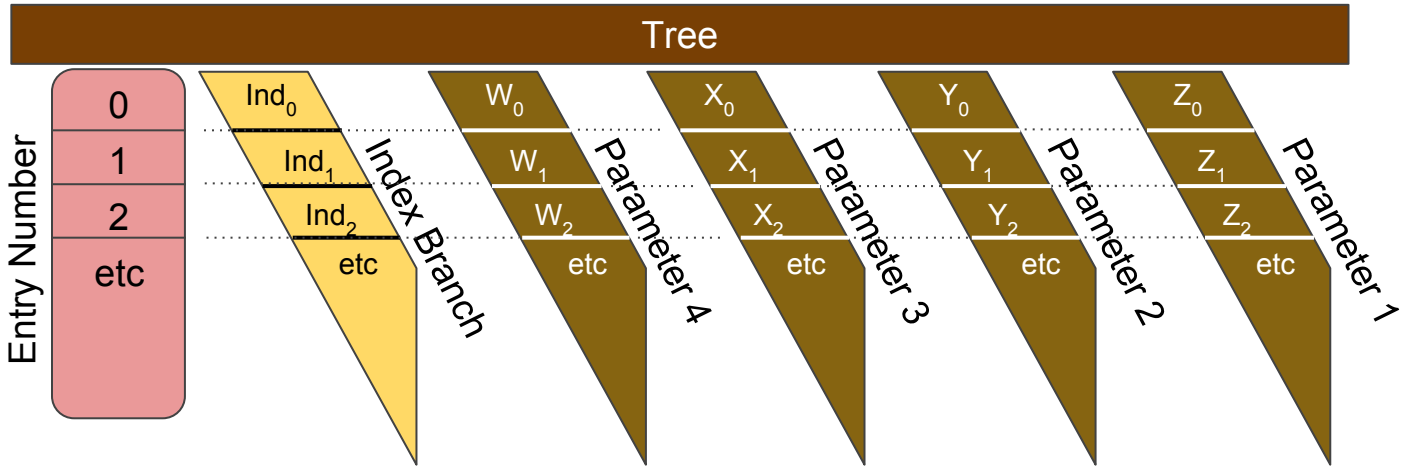


Solution

Create separate index branch with unique identifiers for each entry/event

Added at the same time an event is recorded

Assign branch as index (TTreeIndex class)

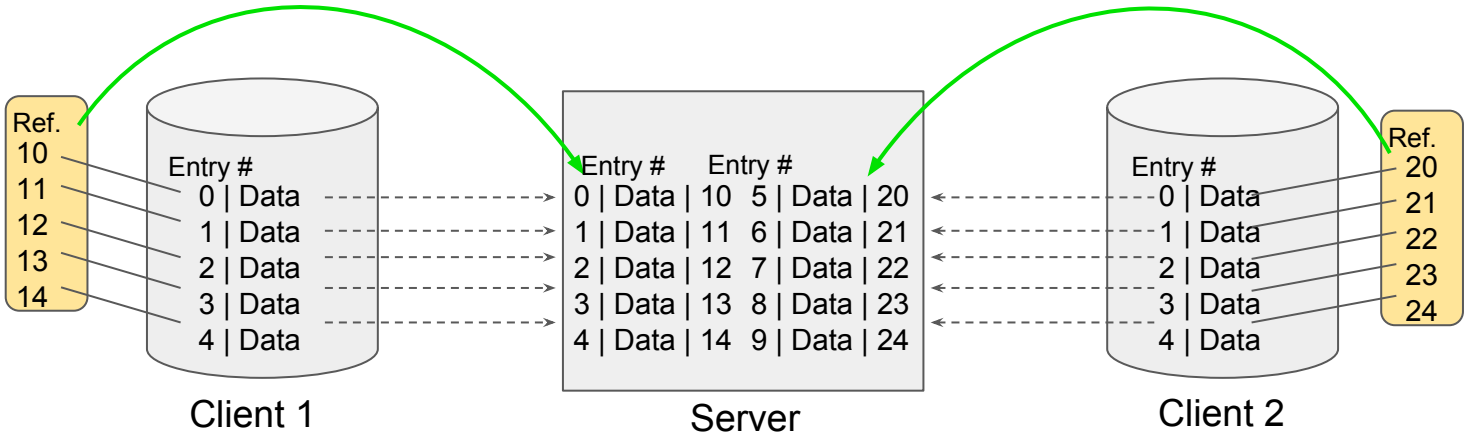


Solution

Each entry has special index number

Server still jumbles order

But entries can be recalled with index number

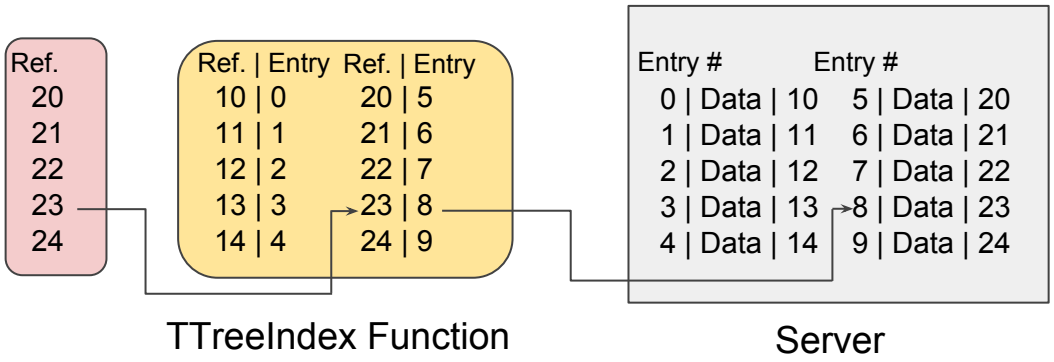


Reading from Index

Tree at server can map index number to entry number

Use index number to get entry number

Call using entry number as before



Athena and POOL

Athena is the data processing framework for ATLAS

- All ATLAS data passes through Athena

POOL: Layer between Athena and storage technology

RootTreeContainer - technology currently used

- Facilitates creation/filling/writing of TTrees and TBranches from input data

Created new technology: RootTreeIndexContainer

- Specialized RootTreeContainer to include indexing

Creating and Filling

POOL has transactional function

Utilize RootTreeContainer version

- Called each time event added to the tree
- Called when writing tree to a file

Index branch created first

Fill index branch each time other branches filled

```
DbStatus RootTreeIndexContainer::transAct(Transaction::Action  action){
  if (action == Transaction::TRANSACT_COMMIT) {
    if (m_tree->GetName()[0]  != '#') {
      if (m_tree == nullptr) return Error;
      if (m_index_foreign== nullptr && m_tree->GetBranch("index_ref")  == nullptr)
{
        m_index_ref = (TBranch*)m_tree->Branch("index_ref",  m_index);
      }
      if (m_index_ref != nullptr) {
        if (RootTreeContainer::size()  > m_index_ref->GetEntries()) {
          *m_index = this->size();
          m_index_ref->SetAddress(m_index);
          if (!m_treeFillMode)  m_index_ref->Fill();
        }
      }
    }
  }
  …
```

Assigning Index Numbers

Creates and assigns index number

Concatenation of entry number and process id

Stock of process ids limited on UNIX systems

Process id resets after a certain number

Bit shifted to keep index number unique

(each bit shifted is equivalent to multiplying the number by a power of 2)

Then entry number is added so that index is sequential

```
long long int RootTreeIndexContainer::nextRecordId()    {
   long long int s = m_index_multi;
   s = s << 32;
   if (m_tree != nullptr) {
      if (m_index_foreign != nullptr) {
         s += m_index_foreign->GetEntries();
      } else {
         m_index_foreign =
(TBranch*)m_tree->GetBranch("index_ref");
         if (m_index_foreign != nullptr) {
            s += m_index_foreign->GetEntries();
         }
      }
   }
   return s;
}
```

Building the Index

Simple matter of declaring branch to be used as index

Called after last fill

(Final time that transAct is called)

Can be called at read, but not preferred

in transAct:

```
if (action == Transaction::TRANSACT_FLUSH) {
        if (m_tree->GetName()[0]  != '#') {
            if (m_tree->GetEntryNumberWithIndex(size())   == -1) {
                m_tree->BuildIndex("index_ref");
            }
        }
}
return Status;
}
```

Reading

Get index number from input

Retrieve entry number

Call original function using entry number

(as things are currently done)

```
DbStatus
RootTreeIndexContainer::loadObject(DataCallBack*  call,
                                   Token::OID_t& oid, DbAccessMode mode) {
  if ((oid.second >> 32) > 0) {
    long long int evt_id = m_tree->GetEntryNumberWithIndex(oid.second);
    if (evt_id == -1) {
      m_tree->BuildIndex("index_ref");
      evt_id = m_tree->GetEntryNumberWithIndex(oid.second);
    }
    if (evt_id >= 0) {
      oid.second = evt_id;
    }
  }
  return RootTreeContainer::loadObject(call, oid, mode);
}
```

Concluding Remarks

This functionality will allow ATLAS to take advantage of newer ROOT developments

eg: Parallel merging (TParallelMergingFile class)

Still undergoing performance testing

Positive results thus far

Has been uploaded to Atlas codebase, but not being used as default