ROOT for Big Data Analysis

Fons Rademakers

Workshop on the future of Big Data management
Imperial College, London, 27-28 June 2013
HEP’s Core Competency

- Handling, processing and analyzing “Big Data”
- For the previous (popular) trends, Grid and Clouds, we were buyers
- For this Big Data trend we have something to offer
- Industry is now reinventing the wheel we started to develop almost 20 years ago as HEP data sizes are becoming more common
Industry Catching Up

• Map-Reduce (Hadoop)
  – Parallel batch solution for analyzing unstructured data
• Dremel (Drill)
  – Interactive analysis of structured nested data stored in columnar format
• SciDB
  – Parallel analysis of matrix data stored in a DB
• Several commercial offerings
“Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and [a novel] columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPU’s and petabytes of data.”
Google’s Dremel

- Dremel: Interactive Analysis of Web-Scale Datasets

“Dremel is a scalable, interactive ad-hoc query system for analysis of read-only nested data. By combining multi-level execution trees and [a novel] columnar data layout, it is capable of running aggregation queries over trillion-row tables in seconds. The system scales to thousands of CPU’s and petabytes of data.”

Sounds pretty much like ROOT
The ROOT System

- ROOT is an extensive data handling and analysis framework
  - Efficient object data store scaling from KB’s to PB’s
  - C++ interpreter
  - Extensive 2D+3D scientific data visualization capabilities
  - Extensive set of data fitting, modeling and analysis methods
  - Complete set of GUI widgets
  - Classes for threading, shared memory, networking, etc.
  - Parallel version of analysis engine runs on clusters and multi-core
  - Fully cross platform, Unix/Linux, Mac OS X and Windows
  - 2.7 million lines of C++, building into more than 100 shared libs

- Development started in 1995

- Licensed under the LGPL
ROOT - In Numbers of Users

- Ever increasing number of users
  - 6800 forum members, 68750 posts, 1300 on mailing list
  - Used by basically all HEP experiments and beyond
Imperial College Workshop 27-28 June 2013

ROOT - In Plots
1000*((sin(x)/x)^2*(sin(y)/y))+200
As of today

177 PB

of LHC data
stored in ROOT format

ALICE: 30PB, ATLAS: 55PB, CMS: 85PB, LHCb: 7PB
The Importance of the C++ Interpreter

• The CINT C++ interpreter is the core of ROOT for:
  – Parsing and interpreting code in macros and on command line
  – Providing class reflection information
  – Generating I/O streamers and columnar layout

• We are moving to a new Clang/LLVM based interpreter called Cling

bash$ root
root [0] TH1F *hpx = new TH1F("hpx","This is the px distribution",100,-1,1);
root [1] for (Int_t i = 0; i < 25000; i++) hpx->Fill(gRandom->Rndm());
root [2] hpx->Draw();

bash$ cat script.C
{
   TH1F *hpx = new TH1F("hpx","This is the px distribution",100,-1,1);
   for (Int_t i = 0; i < 25000; i++) hpx->Fill(gRandom->Rndm());
   hpx->Draw();
}
bash$ root
root [0] .x script.C
ROOT Object Persistency

- Scalable, efficient, machine independent format
- Orthogonal to object model
  - Persistency does not dictate object model
- Based on object serialization to a buffer
- Automatic schema evolution (backward and forward compatibility)
- Object versioning
- Compression
- Easily tunable granularity and clustering
- Remote access
  - HTTP, HDFS, Amazon S3, CloudFront and Google Storage
- Self describing file format (stores reflection information)
- ROOT I/O is used to store all LHC data (actually all HEP data)
ROOT I/O in JavaScript

• Provide ROOT file access entirely locally in a browser
  – ROOT files are self describing, the “proof of the pudding...”
Object Containers - TTree’s

- Special container for very large number of objects of the same type (events)
- Minimum amount of overhead per entry
- Objects can be clustered per sub object or even per single attribute (clusters are called branches)
- Each branch can be read individually
  - A branch is a column
TTree - Clustering per Object

Tree entries

Streamer

Branches

Tree in memory

File
TTree - Clustering per Attribute

Streamers

Object in memory

File
Processing a TTree

TSselector

Begin()
- Create histograms
- Define output list

Process()
- preselection
- analysis
  Ok

Terminate()
- Finalize analysis
  (fitting, ...)

Event

Branch

Leaf

Leaf

Branch

Leaf

Leaf

Branch

Leaf

Loop over events

TTree

1

2

n

last

Read needed parts only

Imperial College Workshop 27-28 June 2013
// Abbreviated version
class TSelector : public TObject {
protected:
    TList *fInput;
    TList *fOutput;
public
    void Notify(TTree*);
    void Begin(TTree*);
    void SlaveBegin(TTree *);
    Bool_t Process(int entry);
    void SlaveTerminate();
    void Terminate();
};
... 

... 

// select event
b_nlhk->GetEntry(entry); if (nlhk[ik] <= 0.1) return kFALSE;
b_nlhpi->GetEntry(entry); if (nlhpi[ipi] <= 0.1) return kFALSE;
b_ipis->GetEntry(entry); ipis--; if (nlhpi[ipis] <= 0.1) return kFALSE;
b_njets->GetEntry(entry); if (njets < 1) return kFALSE;

// selection made, now analyze event
b_dm_d->GetEntry(entry);  //read branch holding dm_d 
b_rpd0_t->GetEntry(entry);  //read branch holding rpd0_t 
b_ptd0_d->GetEntry(entry);  //read branch holding ptd0_d 

//fill some histograms
hdmd->Fill(dm_d);
h2->Fill(dm_d, rpd0_t/0.029979*1.8646/ptd0_d);
... 

...
 RAW - Using SQL to Query TTree's

- Developed at DIAS lab @ EPFL
- SQL makes querying easy

```sql
SELECT event
FROM root:/data1/mbranco/ATLAS/*.root
WHERE
  ( event.EF_e24vhi_medium1 OR event.EF_e60_medium1 OR
  event.EF_2e12Tvh_loose1 OR event.EF_mu24i_tight OR
  event.EF_mu36_tight OR event.EF_2mu13) AND
  event.muon.mu_ptcone20 < 0.1 * event.muon.mu_pt AND
  event.muon.mu_pt > 20000. AND
  ABS(event.muon.mu_eta) < 2.4 AND
  ...
```

- SQL makes querying fast
  - Column-stores & vectorized execution use h/w efficiently.
PROOF - The Parallel Query Engine

• A system for running ROOT queries in parallel on a large number of distributed computers or many-core machines
• PROOF is designed to be a transparent, scalable and adaptable extension of the local interactive ROOT analysis session
• Extends the interactive model to long running “interactive batch” queries
• Uses xrootd for data access and communication infrastructure
• For optimal CPU load it needs fast data access (SSD, disk, network) as queries are often I/O bound
• Can also be used for pure CPU bound tasks like toy Monte Carlo’s for systematic studies or complex fits
The PROOF Approach

- Cluster perceived as extension of local PC
  - *Same macro and syntax* as in local session
- More *dynamic* use of resources
- Real-time feedback
- Automatic *splitting* and *merging*
PROOF - A Multi-Tier Architecture

Adapts to wide area virtual clusters

Geographically separated domains, heterogeneous machines

Optimize for data locality or high bandwidth data server access

Very important

Network performance

Less important
From PROOF

Diagram showing the network setup:
- ROOT Client
- PROOF Master
- xrootd/xpd
- PROOF Worker

Connections:
- TCP/IP
- Unix Socket
- Node
To PROOF Lite

ROOT Client/PROOF Master

PROOF Worker

PROOF Worker

PROOF Worker

Unix Socket

Node
Benchmarking with PROOF-Lite

CPU test

<table>
<thead>
<tr>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD, HDD, SSD</td>
</tr>
</tbody>
</table>

HDD, SSD

Barbone, Donvito, Pompilii CHEP2012

SAS, SSD (CMS data)

Barbone, Donvito, Pompilii CHEP2012

Intel Xeon E7-4870 2.4 GHz
4 sockets, hyper-threading
80 cores, 125 GB RAM
M. Botezatu / OpenLab

Barbone, Donvito, Pompilii CHEP2012
PROOF on Demand

- Use PoD to create a temporary dedicated PROOF cluster on batch resources
- Uses an Resource Management System to start daemons
  - Master runs on a dedicated machine
  - Easy installation
  - RMS drivers as plug-ins: gLite, PanDa, HTCondor, PBS, LSF, OGE
  - ssh plug-in to control resource w/o and RMS
- Each user gets a private cluster
  - Sandboxing, daemon in single-user mode (robustness)
  - Scheduling, auth/authz done by RMS
PROOF on Clouds

- A lot of computing resources available via clouds as virtual machines
- Several PROOF tests has been made on clouds
  - Amazon EC2, Google CE (ATLAS/BNL)
  - Frankfurt Cloud (GSI)
- Dedicated CernVM Virtual Appliance with the relevant services to deploy PROOF on cloud resources
  - PROOF Analysis Facility As A Service
ROOT Roadmap

- Current version is v5-34-05
  - It is an LTS (Long Term Support) version
  - New features will be back ported from the trunk
- Version v6-00-00 is scheduled for when it is ready
  - It will be Cling based
  - It will not contain anymore CINT/Reflex/Cintex
  - GenReflex will come in 6-02
  - It might not have Windows support (if not, likely in 6-02)
  - Several “Technology Previews” will be made available
    - Can be used to start porting v5-34 to v6-00
Conclusions

- HEP has a long experience in Big Data
- We have a number of interesting products on offer
- We should make a concerted effort to better promote and market our products
- EU Big Data projects could be used to extend and document our products for a much wider community