Using Big Data Technologies for HEP Analysis

The CMS Big Data Project:
M. Cremonesi, O. Gutsche, B. Jayatilaka, J. Kowalkowski, S. Sehrish [FNAL]
L. Canali, V. Dimakopoulos, M. Girone, V. Khristenko, E. Motesnitsalis [CERN-IT]
S.-Y. Hoh, J. Pazzini, M. Zanetti [Padova]
P. Elmer, J. Pivarski, A. Svyatkovskiy [Princeton]
I. Fisk [Simons Foundation]
A. Melo [Vanderbilt]
Data Events

- **Particle detection** = record physics quantities (energy, flight path) of particles produced in a collision
  - Quantities measured from the interaction of particles and the different detector components
    - 100 Million individual measurements
    - All measurements of a collision together are called **event**
Event Reconstruction

- Detector signals (and equivalent simulated signals) need to be reconstructed to learn about the particles that produced them.
- The reconstructed events are then used for analysis.
Experimental Particle Physics from Computing Perspective

- Detect particle interactions (data), compare with theory predictions (simulation)
  - Black dots: recorded data
  - Blue shape: simulation
  - Red shape: simulation of new theory (in this case the Higgs)
Detector Data
Simulation
Central

RAW Data ➔ Reconstruction Algorithms ➔ RECO Data ➔ Analysis Software

Chaotic
CMS Data Volume @ HL-LHC

- Extract physics results will require to handle/analyze a lot more data
  - must trim inefficiencies
- Explore industry technologies as suitable candidates for user analysis
CMS Big Data Project

• Group created end of 2015
  - collaboration between FNAL, Diana-HEP, and CERN-IT
  - website: https://cms-big-data.github.io

• Rapidly expanding:
  - Vanderbilt and Padova joined last year

• CERN Openlab enables partnership with industry:
  - CERN Openlab/Intel project called "CMS Data Reduction Facility"
  - Project includes CERN fellow supporting the development and testing of the reduction facility
  - Intel actively taking part in project
  - Sponsoring of CERN fellow included in the project

Thanks to Intel and Cofluent for the support over these years
CHEP 2016: Proof of Principle

• Usability Study using Apache Spark:
  - Analyzer code in Scala
  - Input converted in Avro: [https://github.com/diana-hep/rootconverter](https://github.com/diana-hep/rootconverter)

• Improved user experience with optimized bookkeeping

arXiv:1711.00375
Several technical advancements:

- **stability to read root files in Spark**: [https://github.com/diana-hep/spark-root](https://github.com/diana-hep/spark-root), eliminating the need to convert in a more suitable format

- **Capability to read input files remotely using XRootD** (e.g. from EOS at CERN): [https://github.com/cerndb/hadoop-xrootd](https://github.com/cerndb/hadoop-xrootd), eliminating the need to store files on HDFS
Outline

- Scalability tests and first performance measurements
  - Test the capability to reduce 1 PB of data to 1 TB in less than five hours with the new tools developed by CERN-IT

- Review of real analysis use cases in Apache Spark
  - Tools developed by CERN IT applied at Padova and Vanderbilt to real physics analysis
  - Usability test and current limitation

- What’s next
  - Goal for the next year
Scalability Tests, Infrastructure and Workload

• Spark cluster:
  - analytix @ CERN: shared infrastructure with \(~1300\) cores, 7 TB RAM

• Storage:
  - HDFS and Remote EOS Public/UAT

• Simple physics analysis use case is applied to select events and reduce the datasets
Scalability Test/1

Increasing the input size while maintaining the same amount of resources

Performance for 814 cores in YARN

Initial configuration: 407 executors, 2 vcores per executor, 7 GB per executor
Scalability Test/2

Increasing the resources while maintaining the same input size (for 2:1 vcore-executor ratio)

<table>
<thead>
<tr>
<th>Number of Executors/Cores</th>
<th>Total Memory:</th>
<th>Runtime:</th>
</tr>
</thead>
<tbody>
<tr>
<td>74/148</td>
<td>0.5 TB</td>
<td>81m</td>
</tr>
<tr>
<td>148/296</td>
<td>1 TB</td>
<td>53m</td>
</tr>
<tr>
<td>222/444</td>
<td>1.5 TB</td>
<td>52m</td>
</tr>
<tr>
<td>296/592</td>
<td>2 TB</td>
<td>51m</td>
</tr>
<tr>
<td>370/740</td>
<td>2.5 TB</td>
<td>50m</td>
</tr>
<tr>
<td>444/888</td>
<td>3 TB</td>
<td>50m</td>
</tr>
</tbody>
</table>
Scalability Test/3

Comparing EOS Public, EOS UAT, and HDFS (191, 6, and 38 nodes respectively)

Performance for 20 TB input size

<table>
<thead>
<tr>
<th>Number of Executors/VCores:</th>
<th>Runtime for EOS Public:</th>
<th>Runtime for EOS UAT:</th>
<th>Runtime for HDFS:</th>
</tr>
</thead>
<tbody>
<tr>
<td>111/222</td>
<td>81m</td>
<td>153m</td>
<td>41m</td>
</tr>
<tr>
<td>222/444</td>
<td>52m</td>
<td>146m</td>
<td>35m</td>
</tr>
<tr>
<td>296/592</td>
<td>51m</td>
<td>144m</td>
<td>33m</td>
</tr>
<tr>
<td>407/814</td>
<td>50m</td>
<td>140m</td>
<td>29m</td>
</tr>
</tbody>
</table>
Scalability Test/4

Understanding the bottlenecks - high network load

<table>
<thead>
<tr>
<th>Cores:</th>
<th>EOS Public</th>
<th>EOS UAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>222 vcores</td>
<td>15 Gbytes/s</td>
<td>6 Gbytes/s</td>
</tr>
<tr>
<td>444 vcores</td>
<td>19 Gbytes/s</td>
<td>7.5 Gbytes/s</td>
</tr>
<tr>
<td>592 vcores</td>
<td>21 Gbytes/s</td>
<td>7.5 Gbytes/s</td>
</tr>
<tr>
<td>814 vcores</td>
<td>21 Gbytes/s</td>
<td>7.5 Gbytes/s</td>
</tr>
</tbody>
</table>

Network Average Utilization for 20 TB input size

![Network Utilization Graph](image-url)
Observations

- We can reduce Data with 72 TB/h
  - Current bottleneck: network throughput from remote storage to Spark cluster
  - We measured up to 20 Gigabytes/s throughput to EOS Public
- These results are about a factor 3 from our original goal of reducing 1 PB to 5 hours
  - Reasonably done with more hardware or software optimizations (Work In Progress)
- Workload optimization profited from cooperation with Intel with Intel CoFluent Technology
  - For this particular job, optimal results were obtained at the 2:1 vcore-executor ratio
Analysis Use-case @ Vanderbilt/Padova

Analysis workflow:
- Load standard ROOT files as DataFrames (DFs)
- Open files over XRootD
- Use Spark to transform DFs
- Aggregate DFs into histograms
- Produce plots, tables, etc. from histograms

Tools used:
- Spark-ROOT - ROOT in Spark
- Hadoop-XRootD - XRootD FS support for Hadoop
- Histogrammar - Data aggregation
- Matplotlib - Python-based plotting

Identical physics use cases, using similar strategy, same tools, but different infrastructure
Usability Test

• Make a first-year CS undergraduate student run the workflow
  - No knowledge of physics whatsoever, limited computing knowledge
  - Able to make the Vanderbilt workflow run in one day

• Portability
  - Run the Padova code at Vanderbilt
  - Major showstopper: environment setup

=> need to write sort of a shared library with site configuration towards full generalization
What’s Next: Coffea Development

• Generalized version of the code used so far by Padova/Vanderbilt
  - Fully portable (no configuration issues)
  - Use-case independent (in principle it already is)
• COmpact Framework For Elaborate Algorithms
• Consist in:
  - List of centrally-produced dataset in experiment-specific format needed for analysis => coffeabeans
  - Custom-made version of the experiment software to produce privately datasets in the experiment-specific format => CoffeaGrinder (this step may be needed to add information)
  - List of privately/centrally produced dataset in experiment-specific format => coffeapowder
  - Apache Spark analysis code => CoffeaMaker
  - Reduced datasets/analysis plots => coffeacups
  - Interface with the experiment statistical packages => CoffeaDrinker
Conclusion

• 2016: proof of principle of the usage of big data technologies in HEP analysis
  - Limitation have been identified to set the focus for 2017
• 2017: development of new tools
  - Spark-root, Hadoop-XRootD connector
• 2018: scalability and usability tests, performance measurements
  - Some bottlenecks have been identified
    • Scalability test: need to scale up the Spark infrastructure and possibly the network
    • Usability test: need to generalize the site configuration
• 2019: Coffea development
Backup
Current Analysis Workflow

• Input:
  - Centrally produced output of reconstruction software, reduced content optimized for analysis
  - Apply updated CMS reconstruction recipes
  - Too big for interactive analysis

• Ntupling:
  - Convert into format suited for interactive analysis
  - Still too big for interactive analysis

• Skimming & Slimming:
  - dropping events/branches in a disk-to-disk copy

• Filtering & Pruning:
  - selectively reading events/branches into memory
CMS Data Reduction Facility

- CERN openlab / Intel project
- Demonstrate reduction capabilities producing analysis ntuples using Apache Spark
- Goal: reduce 1 PB input to 1 TB output in 5 hours (CERN Openlab/Intel project)
CHEP 2016: Proof of Principle

- Not changing the analysis workflow, optimizing the bookkeeping
  - Apache spark
  - Analyzer code in Scala
  - Input converted in Avro: https://github.com/diana-hep/rootconverter, stored on the HDFS
Two loops over file entries, parallel jobs in Spark across cluster

// Reference the whole dataset (not individual files)
val mcsample = avrodd("hdfs://path/to/mcsample/*.avro") ← Input

// First pass (and cache for later)
mcsample.persist()
val mc_sumOfWeights = mcsample.map(_.GenInfo.weight).sum ← Sum of Weights for Simulation

// Second pass on data in cluster's memory
val result = mcsample.filter(cuts).map(toNtuple(_, mc_sumOfWeights, mc_xsec)) ← Main Event Selection

// Save as ntuple
result.toDF().write.parquet("hdfs://path/to/mcsample_ntuple") ← Output

Output ntuple is used for analysis e.g: plots, fits, tables

# Bring the ntuple in as a DataFrame
ntuple = spark.read.parquet("hdfs://path/to/mcsample_ntuple") ← Output contains information of:

ntuple.select("mass").show() • Object (e.g. Muon/Jet)
...

Physics plots!
from PandaCore.Tools.minc import *
from re import sub

metTrigger='(trigger&1)==0'
eleTrigger='(trigger&2)==0'
phoTrigger='(trigger&4)==0'

metFilter='metFilter==1 && egmFilter==1'
presel = 'nFatjet==1 && fj1Pt>200 && nTau==0 && SumS(jetPt>30 && jetIso)<2'

cuts = {
    'signal': tAND(metFilter, tAND(presel, 'nLooseLepton==0 && nLooseElectron==0 && nLoosePhoton==0 && pmet>200 && dphipmet>0.4')),
    'mn': tAND(metFilter, tAND(presel, 'nLoosePhoton==0 && nTau==0 && nLooseLepton==1 && looseLeptonIsoTight==1 && abs(looseLeptonPdgId)==13 && pfUnWmag>200 && dphipfUnW>0.4 && m<160')),
    'mom': tAND(metFilter, tAND(presel, 'nLoosePhotons==0 && nTau==0 && nLooseLepton==1 && looseLeptonIsoTight==1 && abs(looseLeptonPdgId)==11 && pmet>50 && pfUnWmag>200 && dphipfUnW>0.4 && m<160')),
    'emm': tAND(metFilter, tAND(presel, 'pfUnWmag>200 && dphipfUnW>0.4 && nLooseElectron==0 && nLoosePhoton==0 && nTau==0 && nLooseMuon==2 && nTightJet>0 && 60<dilepMass && dilepMass<120')),
    'zee': tAND(metFilter, tAND(presel, 'pfUnWmag>200 && dphipfUnW>0.4 && nLooseMuon==0 && nLoosePhoton==0 && nTau==0 && nLooseElectron==2 && nTightJet>0 && 60<dilepMass && dilepMass<120'))
}

for r in ['mn', 'mom', 'emm', 'zee']:
    cuts['W'+r] = tAND(cuts[r], 'isojetNBtags==0')
    cuts[''+r] = tAND(cuts[r], 'isojetNBtags==1')

for r in ['signal', 'emm', 'zee']:
    cuts[''] = tAND(cuts[r], 'isojetNBtags==0')

for r in ['signal', 'ww', 'tt', 'ee', 'nn', 'emm', 'zee']:
    cuts[''] = tAND(cuts[r], 'fj1DoubleCSV>0.75')
    cuts['_fail'] = tAND(cuts[r], 'fj1DoubleCSV<=0.75')

weights = {
    'signal': {'%s*sf_pt*sf_tt*normalizedWeight*sf_lepID*sf_lepIso*sf_lepTrack*sf_eWKV*sf_qcdV*sf_metTrig*sf_btag0',
                'top': '%s*sf_pt*sf_tt*normalizedWeight*sf_lepID*sf_lepIso*sf_lepTrack*sf_eWKV*sf_qcdV*sf_btag1',
                'w': '%s*sf_pt*sf_tt*normalizedWeight*sf_lepID*sf_lepIso*sf_lepTrack*sf_eWKV*sf_qcdV*sf_btag0',
                'z': '%s*sf_pt*sf_tt*normalizedWeight*sf_lepID*sf_lepIso*sf_lepTrack*sf_eWKV*sf_qcdV*sf_btag0',
    }
    'photon': '%s*sf Chapman normalizedWeight*sf_eWKV*sf_qcdV*sf_pho*sf_photrig *sf_qcdV*sf_btag0',
    # add the additional 2-jet kfactor
    weights['signal'] = weights['signal']
    weights['signal_fail'] = weights['signal']
}