Intel Big Data Analytics

CMS Data Analysis with Apache Spark

Viktor Khristenko and Vaggelis Motesnitsalis

12/01/2018

Collaboration Members

Who is participating in the project?

- CERN IT Department (Openlab and IT-DB)
- Fermilab
- The CMS Experiment
- Intel
- DIANA-HEP

Project Description

What are we trying to do?

- Perform High Energy Physics (HEP) Analytics using Industry Standard Big Data Technologies
- Investigate and experiment with new ways to analyze HEP data
- Produce end-to-end solutions for physics analytics

Project Motivation

Why are we doing it?

- Test the feasibility of the industry standard general purpose processing engines for the HEP Data Processing.
- Find methods to reduce time to physics for the PB and EB datasets
- Improve computing resource utilization.
- Educate academy researches (graduate students, postdocs, etc.) in the use of Big Data Technologies
- Open up the HEP field to a larger community of data scientists

HEP Data Processing

What is currently being used by the CMS experiment?

- c++ / python based workflows
- ROOT I/O
- ROOT Histogramming (Aggregating) Functionality
- Batch Processing - Custom Workload Distribution

HEP Data Processing with Apache Spark

How are Apache Spark workflows different?

- scala / python based workflows with JVM as the primary execution environment
- Lazy evaluation and Code Generation per given Query.
- **ROOT I/O for JVM!**
- Easy scale-out of workflows
- No additional boiler plate for managing batches for ML training.

Data Ingestion: spark-root

0.1.16 available on Maven Central!

How do we ingest data into Apache Spark Dataset API?

- **spark-root** - ROOT I/O for JVM
- Extends Apache Spark's Data Source API
- Maps each ROOT TTree to Dataset[Row]
  - A single TTree => Dataset[Row]
- Parallelization = # ROOT files.
- API is uniform all the Data Sources!

Scala

```
// inject the Dataset[Row]
import org.dianahep.sparkroot.experimental._
val df = spark.read.option("tree", <treeName>).root("<path/to/file>")

// pretty print of the schema
df.printSchema
```

```
|-- Particle: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- fUniqueID: integer (nullable = true)
 |    |    |-- fBits: integer (nullable = true)
 |    |    |-- PID: integer (nullable = true)
 |    |    |-- Status: integer (nullable = true)
 |    |    |-- IsPU: integer (nullable = true)
 |    |    |-- M1: integer (nullable = true)
 |    |    |-- M2: integer (nullable = true)
 |    |    |-- D1: integer (nullable = true)
 |    |    |-- D2: integer (nullable = true)
 |    |    |-- Charge: integer (nullable = true)
 |    |    |-- Mass: float (nullable = true)
 |    |    |-- E: float (nullable = true)
 |    |    |-- Px: float (nullable = true)
 |    |    |-- Py: float (nullable = true)
 |    |    |-- Pz: float (nullable = true)
 |    |    |-- PT: float (nullable = true)
 |    |    |-- Eta: float (nullable = true)
 |    |    |-- Phi: float (nullable = true)
 |    |    |-- Rapidity: float (nullable = true)
 |    |    |-- T: float (nullable = true)
 |    |    |-- X: float (nullable = true)
 |    |    |-- Y: float (nullable = true)
 |    |    |-- Z: float (nullable = true)
 |-- Particle_size: integer (nullable = true)
```

Data Processing: CMS Open Data Example

Let’s tackle real collisions data from the CMS Experiment data with Apache Spark?!

- CMS Public 2010 Muonia Dataset
- 100+ top columns (branches)
- Very complicated nestedness
  - AoS of AoS
- Tested on several TBs of data across > 1K input ROOT files

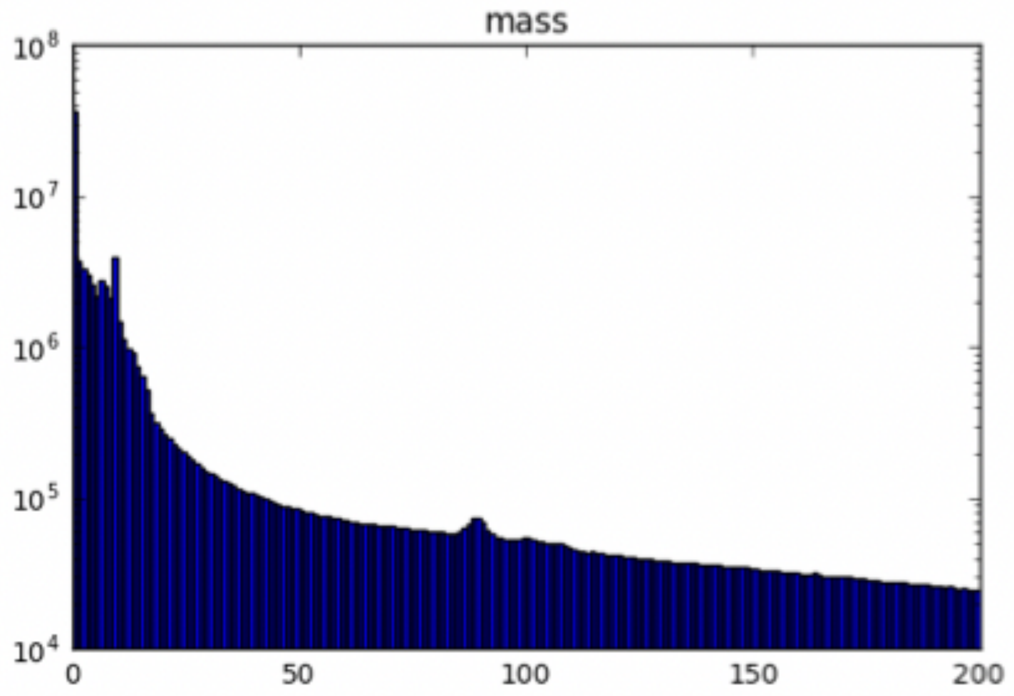
```
|-- patMuons_slimmedMuons__RECO_: struct (nullable = true)
 |    |-- present: boolean (nullable = true)
 |    |-- patMuons_slimmedMuons__RECO_obj: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- m_state: struct (nullable = true)
 |    |    |    |    |-- vertex_: struct (nullable = true)
 |    |    |    |    |    |-- fCoordinates: struct (nullable =
true)
 |    |    |    |    |    |    |-- fX: float (nullable = true)
 |    |    |    |    |    |    |-- fY: float (nullable = true)
 |    |    |    |    |    |    |-- fZ: float (nullable = true)
 |    |    |    |    |-- p4Polar_: struct (nullable = true)
 |    |    |    |    |    |-- fCoordinates: struct (nullable =
true)
 |    |    |    |    |    |    |-- fPt: float (nullable = true)
 |    |    |    |    |    |    |-- fEta: float (nullable = true)
 |    |    |    |    |    |    |-- fPhi: float (nullable = true)
 |    |    |    |    |    |    |-- fM: float (nullable = true)
 |    |    |    |    |-- qx3_: integer (nullable = true)
 |    |    |    |    |-- pdgId_: integer (nullable = true)
 |    |    |    |    |-- status_: integer (nullable = true)
```

CERN openlab

Data Processing: CMS Open Data Example

Let's calculate the invariant mass of a di-muon system?!

- Transform a collection of muons to an invariant mass for each Row (Event).
- Aggregate (histogram) over the entire dataset.




```
# read in the data
df = sqlContext.read\
        .format("org.dianahep.sparkroot.experimental")\
        .load("hdfs:/path/to/files/*.root")

# count the number of rows:
df.count()

# select only muons
muons =
df.select("patMuons_slimmedMuons__RECO_.patMuons_slim
medMuons__RECO_obj.m_state").toDF("muons")

# map each event to an invariant mass
inv_masses = muons.rdd.map(toInvMass)

# Use histogrammar to perform aggregations
empty = histogrammar.Bin(200, 0, 200, lambda row: row.mass)
h_inv_masses = inv_masses.aggregate(empty,
        histogrammar.increment,
        histogrammar.combine)
```

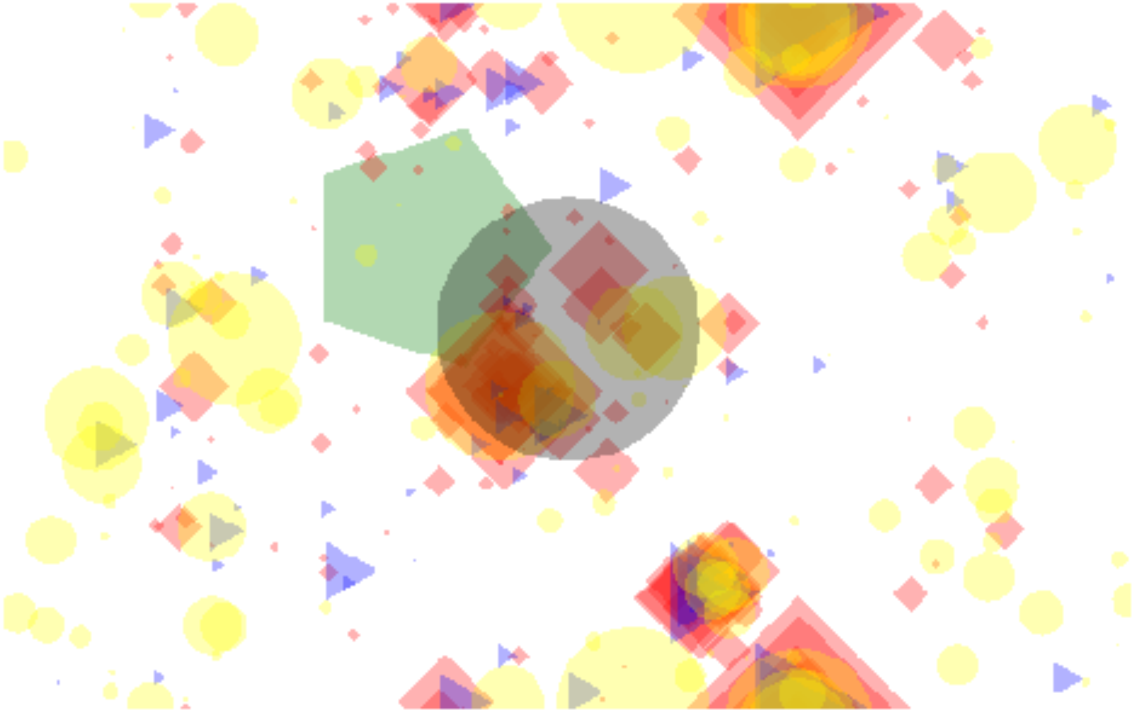
CERN openlab

Data Processing: Feature Engineering

Let's build a feature engineering pipeline for ML Classification using Apache Spark?!

- Simulated Collision Events with:
  - Tracks, Hadrons, Photons, etc.
  - ~10TB of input ROOT files
- Step1: Build a 2D matrix of high level features
- Step2: Build an image
- Step3: Train various classifiers
  - With BigDL / DL4J / mixed solutions
- Step4: Perform Inference
- **All steps are performed using the same Apache Spark Dataset API**

A single image represents a single physics collision



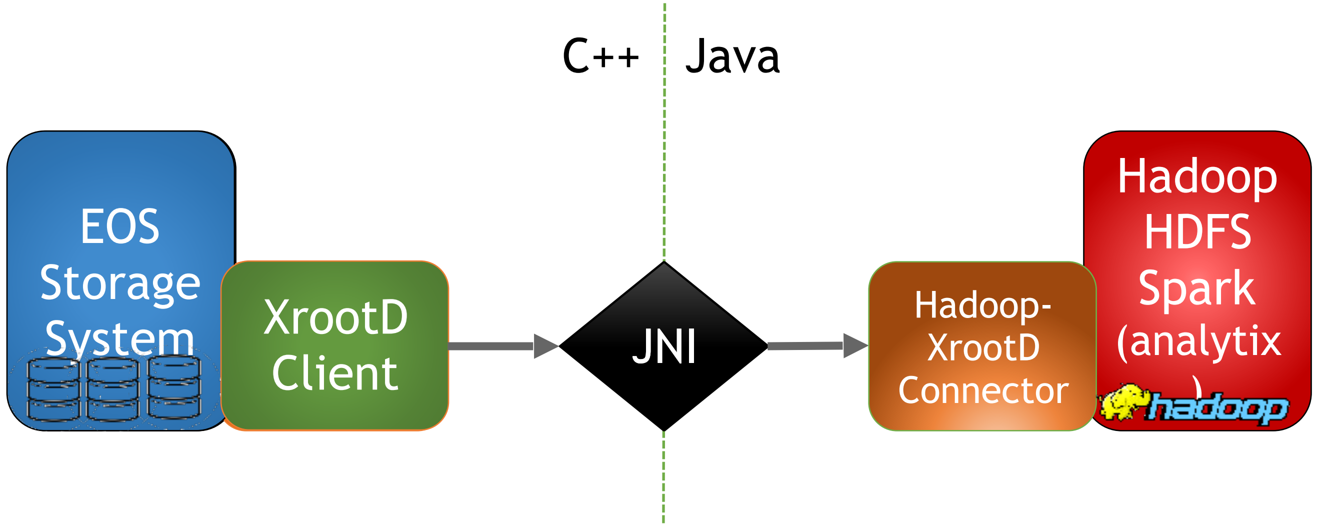
Data Ingestion: EOS vs HDFS

*But what if physics data is on EOS -> **hadoop-xrootd**!*

- “hadoop-XRootD Connector” is a library that connects to the XRootD client via JNI

- It reads files from EOS directly.
  - Avoid copy to/from hdfs!

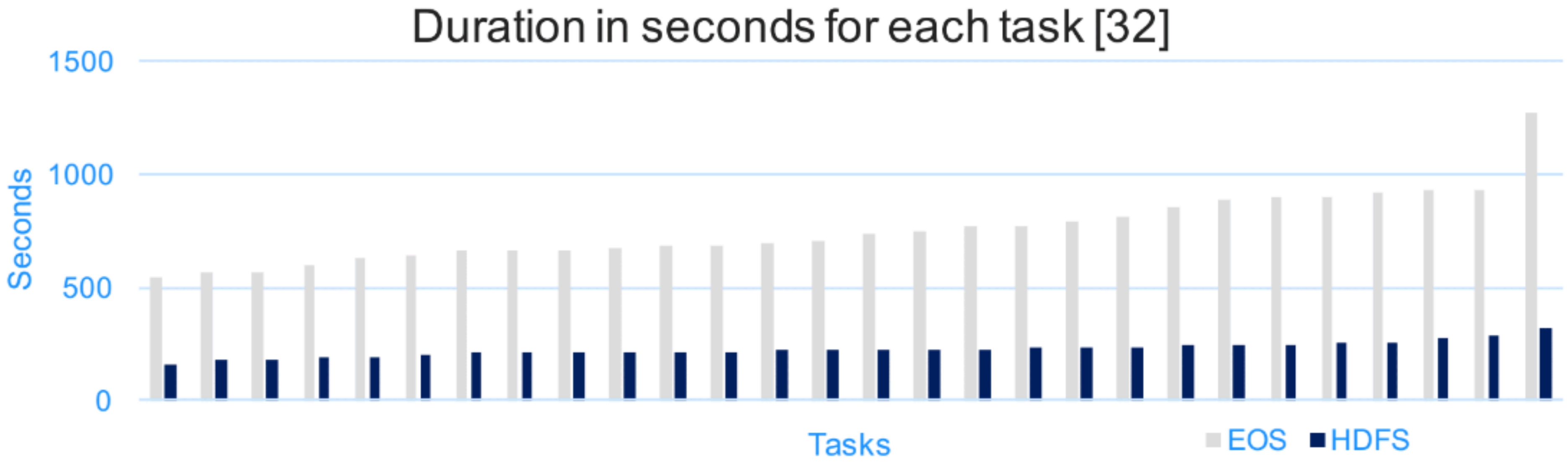
- Soon to be published to GitHub!



CERN openlab

Data Ingestion: EOS vs HDFS

*But what if physics data is on EOS -> **[hadoop-xrootd]()**!*



- Running 2 identical pipelines (input is ~1TB): reading from hdfs vs eos.
- **Reading ROOT files from both file systems works well**
- Throughtput is currently 2-3 times higher reading from hdfs
- Further optimization of the I/O part is necessary

Cluster Infrastructure: CERN Analytix

Where do we run our large scale analyses?

We use the “analytix" Cluster which is provided by the CERN IT Hadoop Service.

Investigating running Apache Spark without Hadoop layer (using kubernetes)

Cluster Characteristics:

- Hadoop Version: 2.6.0-cdh5.7.6
- HDFS Capacity: 4.32 PB
- Cores: ~1200
- Memory: 4.11 TB
- Number of Nodes: 40
- High Availability: Enabled

Recent Talks and Publications

- CMS Analysis and Data Reduction with Apache Spark
  - Proceedings for the 18th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2017)
  - arXiv:1711.00375
- Physics Data Analytics and Data Reduction with Apache Spark
  - 10th Extremely Large Databases Conference
- Status and Plans of the CMS Big Data Project
  - CERN Database Futures Workshop
- More talks and publications -> https://cms-big-data.github.io/pages/pubsntalks.html

General Outlook

A rather personal view on the use of Apache Spark for HEP Data Processing

- Extremely User Friendly!
  - Easy to port python based HEP analyses.
  - Easy to get started
  - Interactive analysis through python/scala shell or jupyter/zeppelin notebooks.
- Easy to scale out your analysis
  - It is just a matter of launching a job on a cluster vs launching locally on a laptop!
- Young Technology and flexible codebase
- Huge user community and adoption in industry
- Scala is a beautiful language! Although python is the right choice for ML.

General Outlook

A rather personal view on the use of Apache Spark for HEP Data Processing

- Apache Spark is optimized for simple tabular schemas.
  - Deeply nested data structures like collection of physics objects -> suboptimal performance.
  - Currently, no means to work efficiently with linear or associative containers

- A lot of parameters have to be optimized for Apache Spark Workflows
  - Garbage Collection Pauses
  - other JVM parameters
  - suboptimal single thread performance w.r.t. c++ based processing

Future Work

How do we plan to move forward?

- We do have ROOT I/O for JVM -> have to improve / optimize / support!
- Experiment with ML Frameworks: Intel BigDL
- Scale out -> investigate the scalability up to 1PB (so far tens of TBs)
- Optimize various workflow specific parameters (Garbage Collection, etc.)
- Investigate the use of Apache Spark on HPC Systems!
- Leverage Intel® CoFluent™ Technology to perform cluster level optimizations!

Questions?

viktor.khristenko@cern.ch

Backup

- spark-root GitHub: https://github.com/diana-hep/spark-root
- histogrammar GitHub: https://github.com/histogrammar/histogrammar-scala
- CMS Big Data Project: https://cms-big-data.github.io