

Employing HPC DEEP-EST for HEP Data Analysis

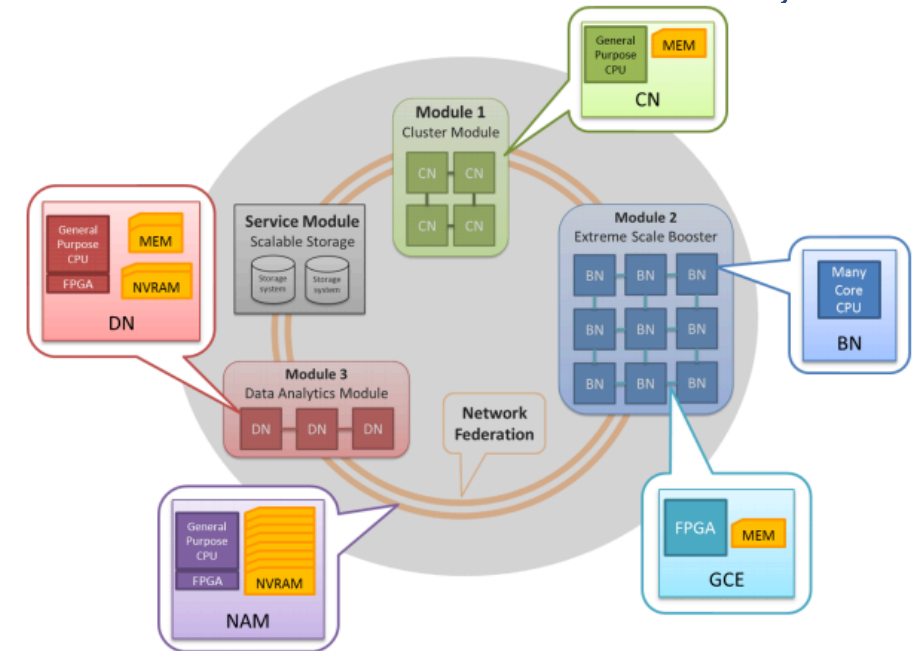
Viktor Khristenko (CERN, DEEP-EST), Maria Girone (CERN)

Outline

- The DEEP-EST Project
- Goals and Motivation
- HEP Data Analysis
 - on HPC
 - with Apache Spark on HPC
- Summary

The DEEP-EST Project

- DEEP - Extreme Scale Technologies.
- R&D for Exascale HPC.
- CERN/CMS is a collaborating partner.
- European Project aiming to build Modular Supercomputing Architecture. Located at Juelich Supercomputing Center (JSC)
- The Project is in the design phase - collecting application requirements and deciding on the overall architecture



Dynamical Exascale Entry Platform - Extreme Scale Technologies

Goals and Motivation for HEP

- Explore conventional HEP analysis pipelines/workflows on HPC infrastructure
 - Experiment with ways to deliver software stack
 - Experiment with new architectures
- Explore (R&D) heterogenous options for data analysis
 - OpenCL / CUDA enabled devices
- Explore the usability of Apache Spark for HEP Data Analysis with HPDA resources
 - HPDA - High Performance Data Analytics

HEP Data Analysis on HPC: Infrastructure

- software stack delivery
 - singularity / docker / shifter containers
 - CernVM-FS
- data locality
 - staging locally to the cluster
 - streaming from remote Storage Elements
- Batch Submission / Resource Provisioning
 - Slurm
- File Systems
 - BeeGFS, GPFS
- Network Attached Memory (NAM)
 - as memory (has its own malloc)
 - as storage/scratch (put/get)
- Global Collective Engine (GCE)
 - FPGA-based MPI collective operations

HEP Data Analysis on HPC: Current Activities

- Benchmarking
 - CMS Simulation + Reconstruction for MC (ttbar)
 - CMS Reconstruction for collision data (different 2017 reco sequences)
- Exploiting heterogenous resources (processing units)
 - OpenCL / CUDA Implementation of RECO parts (e.g. CMS Calorimeters Raw Data Decoding and Local Reco)
 - Batching vs per-Event offload vs Remote offload
 - Understand what a heterogenous framework should look like
- Programming Models
 - OpenCL - a subset of C99, run-time compilation. Soon to come -> **SYCL**
 - CUDA - compile-time Driver and Run Time APIs, possible to reuse source

HEP Data Analysis with Apache Spark on HPC: Overview



- Apache Spark - General Purpose Processing Engine
- Build uniform, high performance pipelines with a single API
- Easy scale-out of workflows
 - from laptop to the cluster
- Use in standalone or utilize Hadoop layer.
 - Optimal for HPC/HEP
 - No need to maintain Hadoop daemons
 - No need to have hdfs - use eos, gpfs, ...



HEP Data Analysis with Apache Spark on HPC: Current Activities



- data ingestion
 - ROOT I/O
- data engineering / analysis / processing
 - Establish baseline benchmarks for various physics queries
 - Utilize TBs of Open Data
- machine learning
 - Employ ML/DL frameworks on top of Apache Spark (e.g. BigDL, Deeplearning4j, TensorFlowOnSpark)



Data Ingestion: spark-root 0.1.16 on Maven Central!



- ROOT I/O for JVM.
 - A completely separate code base. Huge Thanks to ROOT Team: Axel/Danilo/Philippe!
 - There is almost 20-25 years old history of the JVM code base...
 - Tested on TBs of CMS Open Data!
- Extends Apache Spark's Data Source API.
- Represents ROOT TTree as Dataset[Row] upon entry.
 - A single TTree => Dataset[Row]
- Parallelization = # files
- Implementation (Data Source) is modeled after parquet implementation.



Data Ingestion: spark-root 0.1.15 on Maven Central!



- Download spark's tar: <https://spark.apache.org/downloads.html> and unzip
- Start a scala shell:
 - `./bin/spark-shell --packages org.diana-hep:spark-root_2.11:0.1.16`
- Or start a python shell:
 - `./bin/pyspark --packages org.diana-hep:spark-root_2.11:0.1.16`
- Start analyzing/processing
- Straight-forward integration with Jupyter/Zeppelin Notebooks (any other ones..)



HEP Data Analysis with Apache Spark on HPC: Infrastructure



- No Hadoop layer
 - Standalone Spark clusters
 - Investigate Kubernetes
- Integration with the batch system (Slurm)
 - Currently custom scripts to launch a Spark cluster after submitting a slurm job
- Data Locality
 - Investigate streaming data from remote storage elements (CERN EOS)
 - Currently working with data “locally” on GPFS/BeeGFS



HEP Data Analysis with Apache Spark: Biased Outlook



- + **Collaboration**: Apache Spark, **to a large extent**, removes boundaries between experiment-specific analysis frameworks.
- + **Productivity**: Easy to get started -> simple to setup locally -> fast to deploy to a cluster.
- + **Interactivity**: Sufficiently performant to allow real time analysis and exploration over a subset of data.
- - **JVM**: **At this point**, it's non-trivial to optimize for resource-bound processing workflows. (e.g. Garbage Collection...)



Summary

- The DEEP-EST Project - R&D for Exascale HPC
 - R&D opportunity for CERN/CMS for heterogenous computing
 - R&D opportunity for CERN/CMS to exploit HPC infrastructure
- Heterogenous Data Analysis is the future
 - HPC systems become more heterogenous
 - not only from compute but also from memory/storage point of view
 - Need to make use of them efficiently
- General Purpose Solutions - a viable option for HEP Data Analysis?!
 - spark-root - Apache Spark Data Source for ROOT I/O!
 - Apache Spark - very attractive for final stages of the analysis

The DEEP projects DEEP, DEEP-ER and DEEP-EST have received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no ICT-610476 and no ICT-287530 as well as the Horizon2020 funding framework under grand agreement no. 754304.

Backup



Data Processing: CMS Open Data Example

```
# 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_slimme
dMuons__RECO_obj.m_state").toDF("muons")

# map each event to an invariant mass
# inv_masses = muons.rdd.filter(lambda row: row.muons.size==2)
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)
```

