Big Data helps particle physicists to concentrate on science

Matteo Cremonesi(**), Oliver Gutsche(**), Bo Jayatilaka (**), Jim Kowalkowski(**), Cristina Mantilla(**), Jim Pivarski(*), Saba Sehrish(**), Alexey Svyatkovskiy(*)
(*) Princeton University (**) Fermilab

Abstract

In this poster, we evaluate Apache Spark for High Energy Physics (HEP) analyses using an example from the CMS experiment at the Large Hadron Collider (LHC) in Geneva, Switzerland. HEP deals with the understanding of fundamental particles and the interactions between them and is a very compute- and data-intensive statistical science. Our goal is to understand how well this technology performs for HEP-like analyses. Our use case focuses on searching for new types of elementary particles explaining Dark Matter in the universe. We provide different implementations of this analysis workflow; one using Spark on the Hadoop ecosystem, and the other using Spark on high performance computing platforms. The analysis workflow uses official experiment data formats as input and produces publication level physics plots. We compare the performance and productivity of the current analysis with the two above-mentioned approaches and discuss their respective advantages and disadvantages.

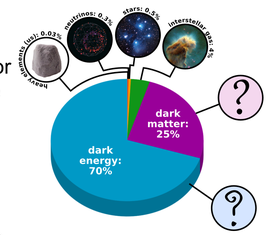
Introduction to the CMS Dark Matter search

In a particle collision, Dark Matter would be produced in association with visible particles.
Dark Matter particle(s) would propagate through the detector undetected while visible particles would leave signals in the CMS detector.
The signature we search for in Dark Matter production at CMS is an energy imbalance, or "missing transverse energy" associated with detectable particles.

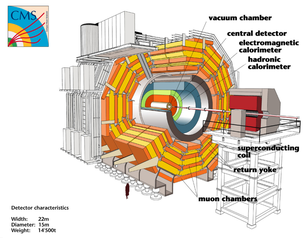
- This signature is commonly referred to as "monoX" where "X" can be a light quark or gluon, a vector boson, or a heavy quark such as a bottom or top quark.
- Focus our search on the "monoTop" signature, where the detectable particle is a top quark.

Challenges of the analysis:

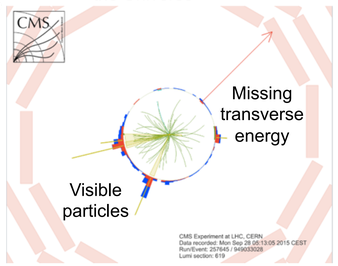
- Top quarks are relatively rare, we need to identify collisions producing top quarks with high efficiency
  - Advanced computational techniques such as artificial neural networks and boosted decision trees can greatly improve the efficiency of the top identification process
- Large backgrounds stemming from known processes will still dominate any present signal
  - Optimize the collision selection via advanced computational techniques



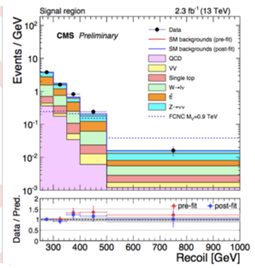
Matter-Energy Composition of the Universe



The CMS detector



Example of selected collision



Publication physics plot

Problem description/Solution using ROOT

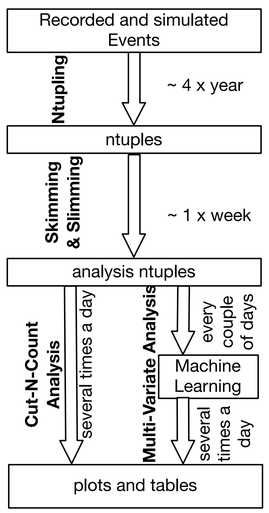
Reconstructing and analyzing HEP collisions

- Physics analyses require collisions both recorded by the detector and simulated using Monte Carlo techniques
- A C++ software framework is used to reconstruct recorded and simulated detector signals, based on a statistics toolkit also uses to persist objects in files
- Reconstructed objects represent particles in a collision (called event) and are used in the analysis step
- Recorded data and simulations are processed centrally and the output is analyzed by all physicists looking for a wealth of different physics signals

Dark Matter analysis use case

1. **N-tupling**
   Input
   - Recorded events: Analysis Object Data (AOD) event format (400kB/event). C++ objects describing the analysis products
   - 5 x 10^8 simulated events for backgrounds and signal (200 TB)
   - Processing time: 2.8 x 10^4 CPU hours

   Output ROOT files in custom Bacon format (2 TB), event and physics objects information stored in vectors and float types, and fixed types of particles per event (Electron, Muon, Tau, etc)
2. **Skimming (reduce number of events) and Slimming (reduce event content)**
   - Apply analysis pre-selection
   - Output: flat ROOT n-tuples with only necessary information
3. **Final analysis is performed using the flat ROOT ntuples**
   - Output: publication grade plots and tables



Using Spark

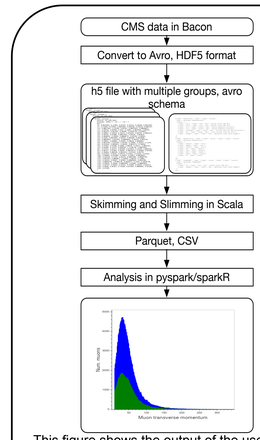
Spark on NERSC
Apache Spark 2.0 is available on Cori and Edison at NERSC.
Edison is used in the initial development and testing; a Cray XC30, with a peak performance of 2.57 petaflops/sec, 133,824 compute cores, 357 terabytes of memory, and 7.56 petabytes of disk. Store data in GPFS.
Input data: Convert the n-tuple format (ROOT TTree) to the HDF5 format.

- One HDF5 group per particle type e.g. Tau, electron, Muon (ROOT TBranch)
- One HDF5 dataset per particle property in each group e.g. phi, pt (ROOT TLeaf)
- Column-oriented data for faster access
- Custom HDF5 reader to read in a HDF5 group with several 1D datasets into Spark DataFrame

Spark operations and APIs: perform skimming and slimming on Spark DataFrames

- Use map, flatMap and filter transformations.
- Use SQL queries and UDF
- Intermediate results stored in CSV files (smaller dataset)

Statistical analysis and plotting: Read intermediate data into Spark Dataframes or Datasets, and make plots using R and the distributed histogrammar package.



This figure shows the output of the use case workflow. It shows Muon transverse momentum distributions for di-boson simulated samples.

Spark *on* Hadoop
Apache Spark on an SGI Hadoop Linux cluster consisting of 6 data nodes and 4 service nodes all with Intel Xeon CPU E5-2680 v2 @ 2.80GHz, each worker node having 256 GB of memory. Configure the Hadoop cluster without single points of failure using two separate machines as name nodes. Deploy Spark applications using YARN resource manager, store data in HDFS.
Input data: Develop a library to convert ROOT Ttrees (the most common format in HEP) to Apache Avro row-based format readable by Spark and stored in HDFS
Spark operations and APIs: perform skimming and slimming on RDDs

- Use Spark's map, flatMap and filter transformations.
- Use modern Dataset and Dataframe APIs taking advantage of advanced Catalyst query optimizer and direct operations on serialized data, available in Spark 2.0.
- Use Apache Parquet columnar format to store intermediate results between stages of the calculation in HDFS, allowing to easily ingest data into Dataframes and Datasets

Statistical analysis and plotting: Use the distributed histogrammar package

histo·*grammar* /histō,'gɹæm.ər/

Histogrammar is a grammar of composable histogram pieces that together provide all of the capabilities expected in a HEP histogram aggregation package. These pieces, called "primitive aggregators," are all linear monoids, so any composition of them can be filled in any order on any partitioning of the data. We can therefore use the entire Spark cluster for data exploration and final plots without being limited to a few plot types. Histogrammar is only an aggregation tool– we linked it to Bokeh to draw the plots. See http://histogrammar.org for more.

Performance at NERSC using 140 cores, and total of 46 GB compressed data in 78 HDF5 files

- Count the number of events => ~1sec, 3.7 million events
- Sum of weights for 3.7 million events => 2 sec
- Creating and saving files after filtering/cuts => 1-6 min

Similar test was repeated with one uncompressed file with the same data, total 86 GB, initial reading and caching time is twice as long. The rest of the tests have comparable outcome.

Performance at Princeton BD cluster using 140 cores

- Sum of weights for events => 14 sec

Conclusion The goal of our exploratory study is to shorten *time to physics* using Spark

- We have observed good scalability, task distribution and data partitioning
- Availability of pySpark and SparkR high level APIs are appealing to the HEP user community
- Encoding skimming workflow using Scala best practices is challenging along with the optimal use of Spark DataFrame features
- Documentation, and error reporting needs to be better

Future direction

- Improve the HDF5 reader interface
- Optimize skimming workflow
- Scale up to greater than TB data set

Acknowledgement This study is supported by the Department of Energy (DOE). The CMS collaboration is supported by DOE, NSF, and international funding agencies. Histogrammer is supported through the DIANA project by NSF.