IT Information Technology Department

PHYSICS DATA ANALYSIS AND DATA REDUCTION AT SCALE WITH APACHE SPARK

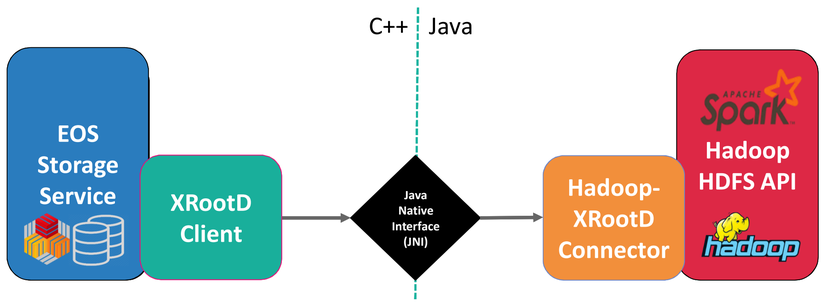
E. Motesnitsalis[1], V. Khristenko[1], M. Migliorini[1], R. Castellotti[1], L. Canali[1], M. Girone[1], D. Olivito[1], M. Cremonesi[2] , J. Pivarski[3], O. Gutsche[2]

[1)]CERN, Geneva, Switzerland; [2)]Fermilab, Batavia, USA ; [3)]Princeton University

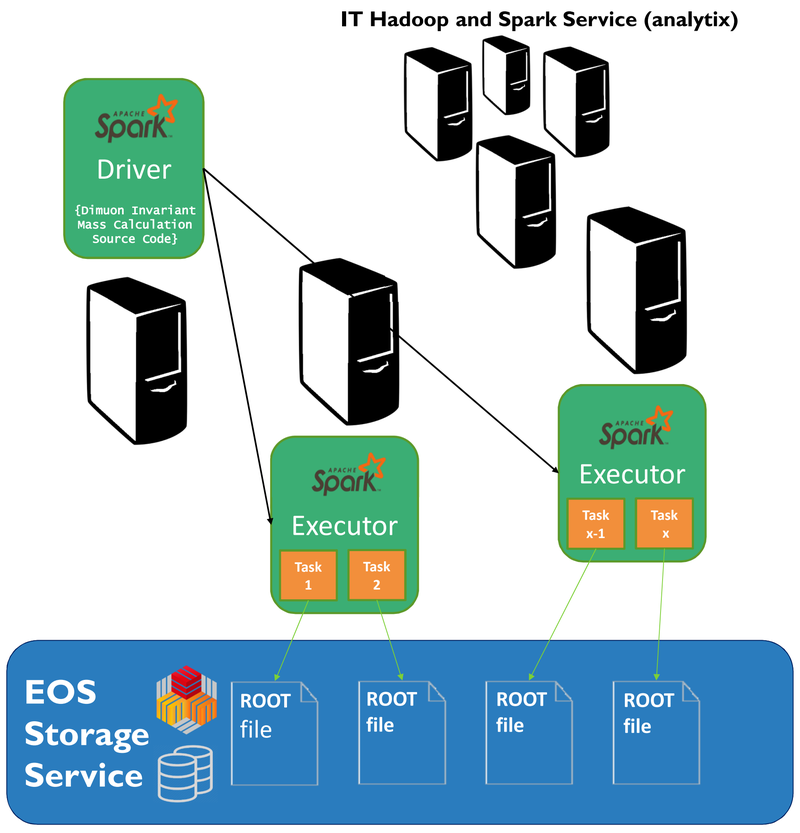
GOALS AND MOTIVATIONS

- The main goal of the project is to perform Data Analysis and Data Reduction at scale using Big Data Technologies over Physics data acquired by CMS and made public on the CERN open data portal
- We are interested in investigating new ways to analyse physics data and allow further development with Streaming & Machine Learning workloads
- We want to adopt new technologies widely used in the industry with modern APIs, development environments and platforms (notebooks etc.)
- This opens data processing for High Energy Physics (HEP) to a larger community of data scientists and data engineers, bringing together domain experts from industry and academia

Hadoop – XRootD Connector Architecture



Test Workload Architecture and File-Task Mapping



CURRENT PROCEDURES AND MILESTONES

- Until today, the vast majority of high energy physics analysis is done using the ROOT Framework processing physics data stored in ROOT format files
- To use big data tools, we have solved two key data engineering challenges:
  1. Read files in ROOT Format using Spark
  2. Access files stored in EOS directly from Hadoop/Spark
- This enabled us to produce, and optimize Physics Analysis Workloads with input up to 1 PB. The Spark infrastructure is now used by several physics analysis groups

SPARK-ROOT AND HADOOP-XROOTD CONNECTOR

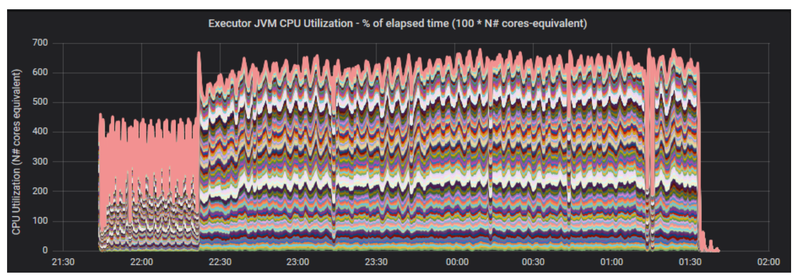
- Spark-root is an open source Scala library which can read ROOT TTrees, infer their schema and import them to Spark Dataframes
- Hadoop-XRootD Connector is an open source Java library that connects to the XRootD client via Java Native Interface
- A parameterized “readAhead” buffer is used to improve performance when reading files from the EOS Service

RESULTS FROM SCALABILITY TESTS

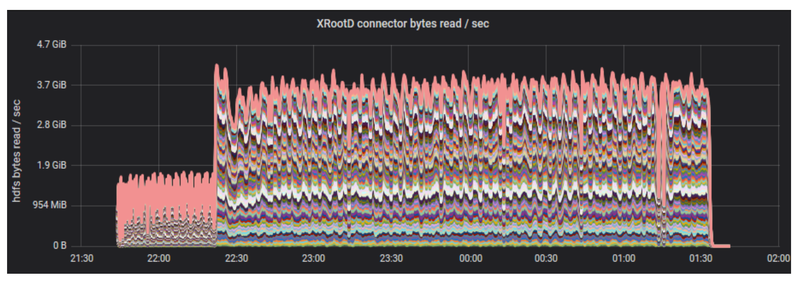
- The data processing job of this project performs event selection (i.e. Data Reduction) and then computes the dimuon invariant mass
- On a single thread/core and one single file as input, the workload reads one branch and produces the calculations in approximately 10 mins for a 4GB file
- The produced results of running at scale are displayed in Graphs 1-5 and Table 1.

Metric Name	Total Time Spent (Sum Over Executors)	Percentage (Compared to Execution Time)
Total Execution Time	**~3000 - 3500 hours**	**1**
CPU Time	~1200 hours	40%
EOS Read Time	~1200 - 1800 hours, depending on readAhead size	40-50%
Garbage Collection Time	~200 hours	7-8 %

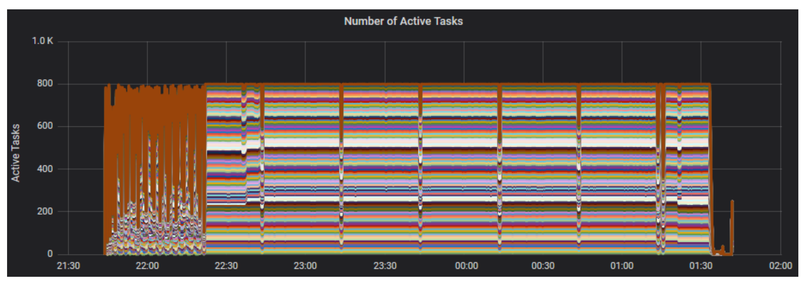
Table 1: Key workload metrics and time spent, measured with Spark custom instrumentation for 1 PB of input with 100 Spark executors, 800 logical cores, 8 logical cores per Spark executor, and variable “readAhead” size between 16 KB and 64 KB.



Graph 3: Executor CPU Usage throughout job execution for 1 PB of input with 64 KB of “readAhead” buffer, 100 Spark executors, and 8 logical cores per executor.

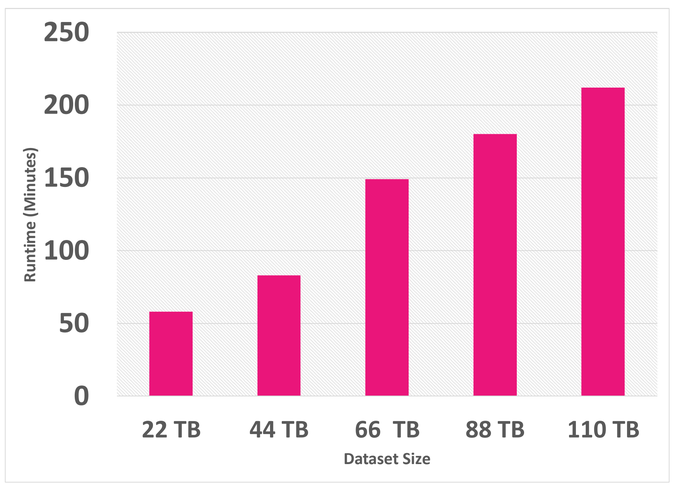


Graph 4: Read Throughput throughout job execution for 1 PB of input with 64 KB of “readAhead” buffer, 100 Spark executors, and 8 logical cores per executor.

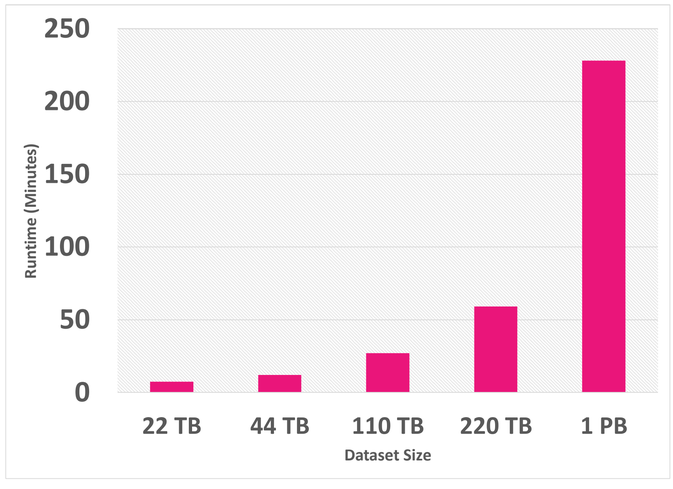


Graph 5: Number of concurrent active tasks throughout job execution for 1 PB of input with 64 KB of “readAhead” buffer, 100 Spark executors, and 8 logical cores per executor.

Performance Results at Scale



Graph 1: Runtime performance in minutes for different input size with 407 Spark executors, 2 cores per Spark executor, 7 GB per Spark executor. The “readAhead” connector buffer is set to 32 MB.



Graph 2: Runtime performance in minutes for different input size with 100 Spark executors, 8 cores per Spark executor, 7 GB per Spark executor. The “readAhead” connector buffer is set to 64 KB which drastically improved the performance compared to Graph 1.

EOS Service

A disk-based, low-latency storage service with a highly-scalable hierarchical namespace, which enables data access through the XRootD protocol. It provides storage for both physics and user use cases via different service instances such as EOSPUBLIC, EOSCMS etc.

FUTURE STEPS

- Repeat the workload scalability tests on top of virtualized/containerized infrastructure with Kubernetes in larger infrastructure and public clouds
- Extend the features of the “Hadoop-XRootD Connector” library (i.e. write to EOS, better packaging, monitoring etc.)
- Extend the workloads to different and more complex use cases of Physics Data Processing as well as use cases for Machine Learning and Online Data Processing (Streaming)

CONCLUSIONS

- We have solved two important data engineering challenges:
  - Hadoop-XRootD Connector can directly access files from the EOS Service
  - Spark-root can read ROOT files and infer their schema into Spark
- Did we achieve the project milestone of reducing 1 PB in 5 hours?
  - Yes, we even dropped to below 4 hours in our latest tests
- Through this project we achieved:
  - Efficient and fast processing of physics data
  - Connecting Libraries between Big Data Technologies and HEP Tools
  - Adoption of Big Data Technologies by CMS physics groups (e.g. University of Padova, Fermilab)