



Physics Data Analytics and Data Reduction with Apache Spark

Evangelos Motesnitsalis

11 October 2017

Contents

1. Goal
2. Motivation and Vision
3. Collaboration Members
4. Standard HEP Analysis Procedures
5. Progress
 - Read Files in ROOT Format directly from Spark
 - Access files stored in EOS directly from Spark/Hadoop
6. Planned Architecture Overview
7. Demo
8. Future Steps
9. Summary and Outlook

Goal

Goal

*“to perform Physics Analysis and Data Reduction
with Big Data Technologies
over data acquired from the CMS Experiment”*

Motivation and Vision

Why Data Analytics and Reduction with Spark?

Motivation and Vision

Why Data Analytics and Reduction with Spark?

- Investigate new ways to analyse physics data and improve resource utilization and time-to-physics

Motivation and Vision

Why Data Analytics and Reduction with Spark?

- Investigate new ways to analyse physics data and improve resource utilization and time-to-physics
- Adopt new technologies widely used in the industry
 - Open the HEP field to a larger community of data scientists
 - Improve chance of researchers on the job market outside academia

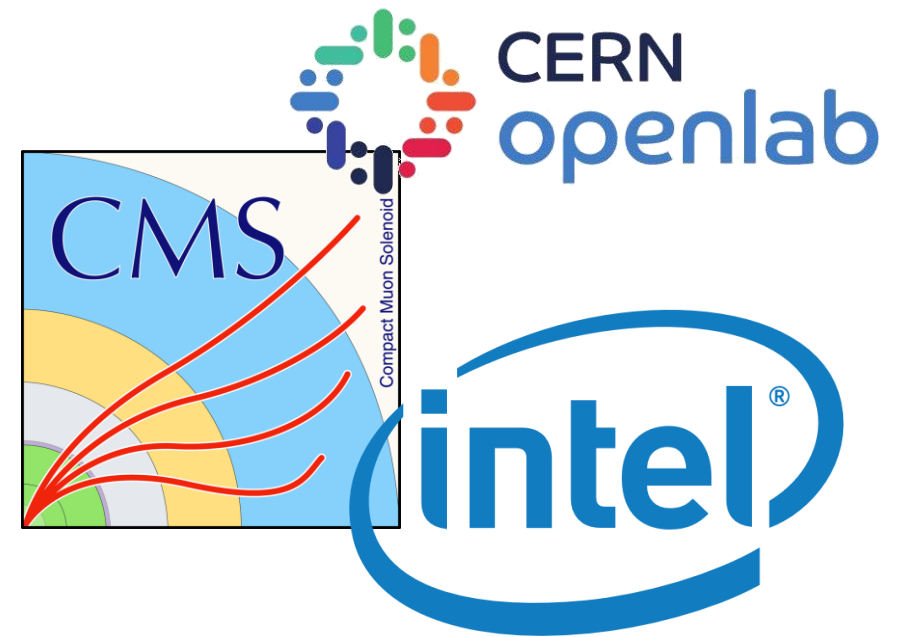
Collaboration Members

Who is participating?

Collaboration Members

Who is participating?

- CERN IT Department (openlab and IT-DB)
- The CMS Experiment
- Intel



Standard HEP Analysis Procedures

How is Physics Analysis done currently?

Standard HEP Analysis Procedures

How is Physics Analysis done currently?

Until today, the vast majority of high energy physics analysis is done with the ROOT Framework which uses physics data that are stored in ROOT format files, within the EOS Storage Service of CERN.

Standard HEP Analysis Procedures

How is Physics Analysis done currently?

Until today, the vast majority of high energy physics analysis is done with the ROOT Framework which uses physics data that are stored in ROOT format files, within the EOS Storage Service of CERN.

ROOT Data Analysis Framework

A modular scientific software framework which provides all the functionalities needed to deal with big data processing, statistical analysis, visualization and file storage. It is mainly written in C++ but also integrated with Python and R.



Standard HEP Analysis Procedures

How is Physics Analysis done currently?

Until today, the vast majority of high energy physics analysis is done with the ROOT Framework which uses physics data that are stored in ROOT format files, within the EOS Storage Service of CERN.

ROOT Data Analysis Framework

A modular scientific software framework which provides all the functionalities needed to deal with big data processing, statistical analysis, visualization and file storage. It is mainly written in C++ but also integrated with Python and R.



EOS Service

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



Progress

What has been done to date?

Progress

What has been done to date?

Two Main Challenges have been solved:

1. Read files in ROOT Format using Spark
2. Access files stored in EOS directly from Hadoop/Spark

Progress

What has been done to date?

Two Main Challenges have been solved:

1. Read files in ROOT Format using Spark
2. Access files stored in EOS directly from Hadoop/Spark

We now have fully functioning Analysis & Reduction examples tested over CMS Open Data (1 TB)

1. Read Files in ROOT Format directly from Spark

DIANA-HEP developed “spark-root”

A Scala library which implements DataSource for Apache Spark

Spark can read ROOT TTrees and infer their schema

Root files are imported to Spark Dataframes/Datasets/RDDs

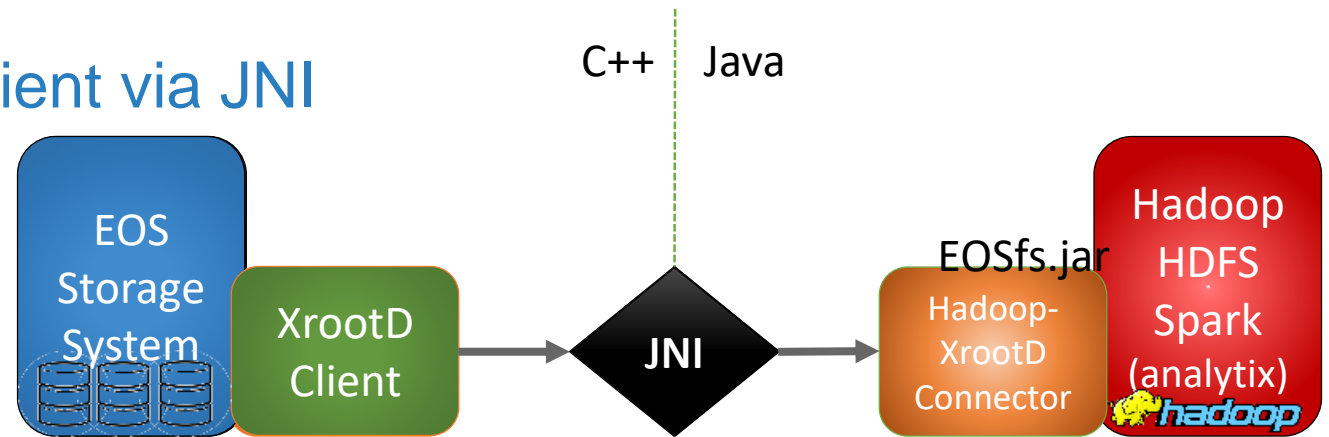


2. Access files stored in EOS directly from Spark/Hadoop

We developed “Hadoop-XrootD Connector”

A library that connects to the XrootD client via JNI

It reads files on EOS directly
(no need for import/export)



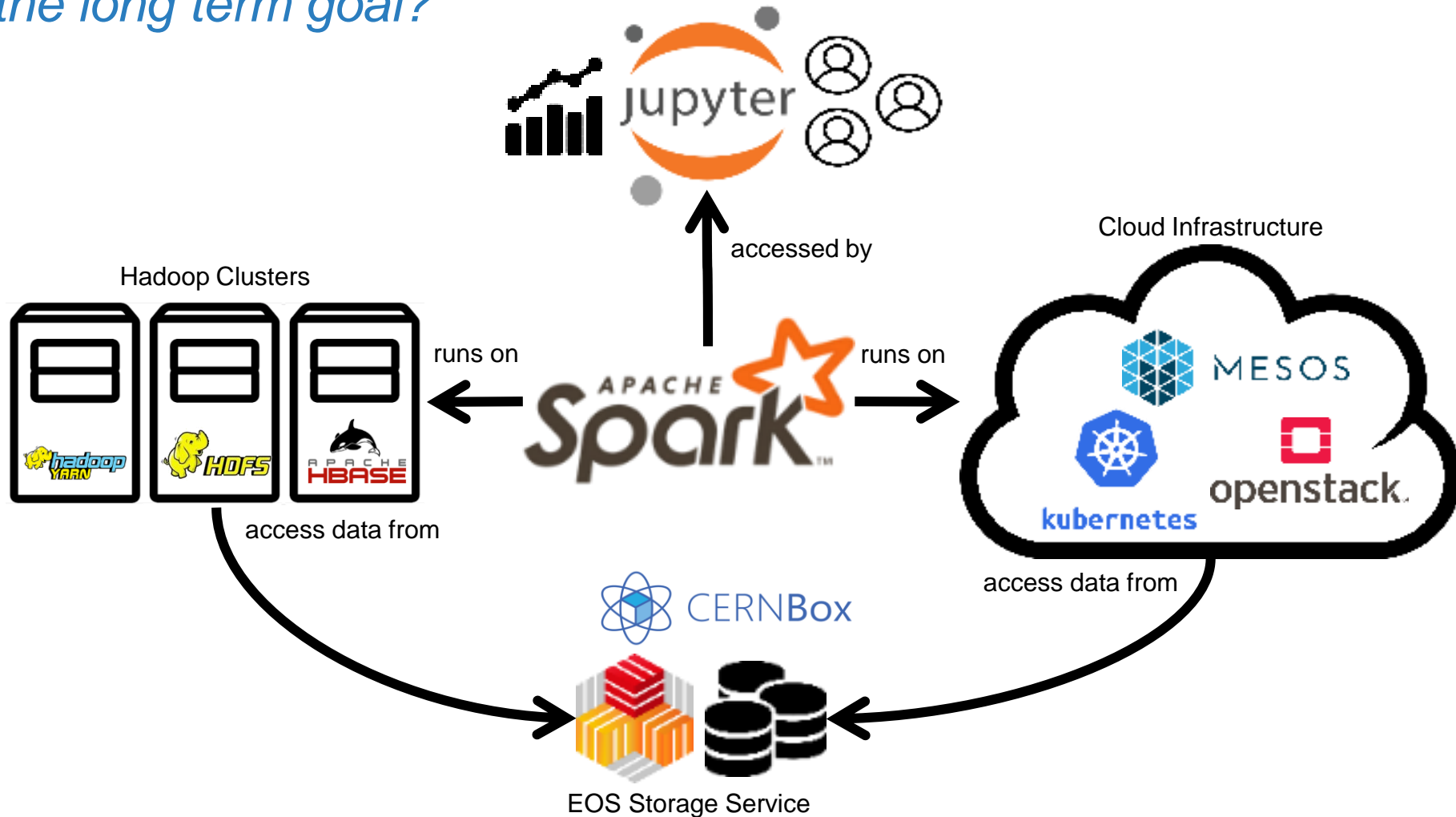
Currently working with 30% the speed of direct HDFS access for a READ operation

Planned Architecture Overview

What is the long term goal?

Planned Architecture Overview

What is the long term goal?



Lightning Demo



physics_analysis_using_swan_spark_template Last Checkpoint: 7 minutes ago (unsaved changes)



Control Panel

Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 2

File Edit View Insert Cell Kernel Help Not Trusted Python 2



Integration of SWAN with Spark clusters

This notebook demonstrates the functionality provided by a SWAN prototype machine that allows to offload computations to an external Spark cluster. The Spark version we are going to use is 2.1.0 and we are going to connect to the analytix cluster (as previously selected in the SWAN web form).

Step 1 - Acquire the necessary credentials to access the Spark cluster.

```
In [1]: import getpass
import os, sys, re

print("Please enter your password")
ret = os.system("echo \"%s\" | kinit" % re.escape(getpass.getpass()))

if ret == 0: print("Credentials created successfully")
else: sys.stderr.write('Error creating credentials, return code: %s\n' % ret)

Please enter your password
.....
Credentials created successfully
```

Lightning Demo

The screenshot shows a Jupyter Notebook titled "physics_analysis_using_swan_spark_template" with a last checkpoint of 9 minutes ago. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help), a toolbar with icons for file operations and code execution, and a status bar indicating "Not Trusted" and "Python 2".

The notebook content is as follows:

Please enter your password
.....
Credentials created successfully

Step 2 - Import the spark libraries

```
In [2]: from pyspark import SparkConf, SparkContext
        from pyspark.sql import SparkSession
```

Step 3 - Notebook plumbing until EOS connector is integrated with SWAN

```
In [3]: !wget -q -O /tmp/eosConnect.zip https://cernbox.cern.ch/index.php/s/6hMOACLErPoLvIR/download
        !unzip -q -o /tmp/eosConnect.zip -d /tmp
```

Step 4 - Build the spark configuration object and create spark context

```
In [4]: conf = SparkConf()
        conf.set('spark.driver.host', os.environ['SERVER_HOSTNAME'])
        conf.set('spark.driver.port', os.environ['SPARK_PORT_1'])
        conf.set('spark.fileserver.port', os.environ['SPARK_PORT_2'])
        conf.set('spark.blockManager.port', os.environ['SPARK_PORT_3'])
        conf.set('spark.ui.port', os.environ['SPARK_PORT_4'])
        conf.set('spark.master', 'yarn')
        conf.set('spark.driver.memory', '4g')
        # following config should be integrated into SWAN
        conf.set('spark.yarn.dist.files', 'file:///tmp/krb5cc_' + os.environ['USER_ID'] + '#krbcache')
        conf.set('spark.executorEnv.KRB5CCNAME', 'FILE:$PWD/krbcache')
        conf.set('spark.driver.extraClassPath', '/tmp/EOSfs.jar')
        conf.set('spark.jars', '/tmp/org.diana-hep_spark-root_2.11-0.1.11.jar,/tmp/org.diana-hep_root4j-0.1.6.jar,/tmp/org.apache.bcel')
        conf.set('spark.executor.extraClassPath', 'file:/tmp/org.diana-hep_spark-root_2.11-0.1.11.jar,file:/tmp/org.diana-hep_root4j-0.1.6.jar')
        conf.set('spark.driver.extraLibraryPath', '/tmp')

        sc = SparkContext(conf = conf)
        spark = SparkSession(sc)
```

Step 5 - Test snippet to check EOS access

```
In [5]: df = spark.read.format("org.dianahep.sparkroot").option("tree", "Events").load("root://eospublic.cern.ch/eos/opendata/cms/Run20")
        df.count()
```

Lightning Demo

physics_analysis_using_swan_spark_template Last Checkpoint: 11 minutes ago (autosaved) Control Panel Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 2

Step 5 - Test snippet to check EOS access

```
In [5]: df = spark.read.format("org.dianahep.sparkroot").option("tree", "Events").load("root://eospublic.cern.ch/eos/opendata/cms/Run20")
df.count()
```

Out[5]: 16590

```
In [6]: data_directory = "root://eospublic.cern.ch/eos/opendata/lhcb/AntimatterMatters2017/data/"
sim_data_df = spark.read.format("org.dianahep.sparkroot").load(data_directory + "PhaseSpaceSimulation.root")
sim_data_df.createOrReplaceTempView("sim_data")
sim_data_df.cache()
sim_data_df.count()
```

Out[6]: 50000

```
In [7]: # Display the first 10 rows in the sim_data_df Spark DataFrame
sim_data_df.limit(10).toPandas() # use pandas only for visualization
```

Out[7]:

	B_FlightDistance	B_VertexChi2	H1_PX	H1_PY	H1_PZ	H1_ProbK	H1_ProbPi	H1_Charge	H1_IPChi2	H1_jsMuon	...	H2_IPChi2	H2_isMuon	H3_
0	0.0	1.0	3551.84	1636.96	23904.14	1.0	0.0	-1	1.0	0	...	1.0	0	36100
1	0.0	1.0	-2525.98	-5284.05	35822.00	1.0	0.0	1	1.0	0	...	1.0	0	-8648
2	0.0	1.0	-700.67	1299.73	8127.76	1.0	0.0	-1	1.0	0	...	1.0	0	-13483
3	0.0	1.0	3364.63	1397.30	222815.29	1.0	0.0	1	1.0	0	...	1.0	0	1925
4	0.0	1.0	-581.66	-1305.24	22249.59	1.0	0.0	-1	1.0	0	...	1.0	0	-2820
5	0.0	1.0	112.84	-13297.98	51882.87	1.0	0.0	1	1.0	0	...	1.0	0	-440
6	0.0	1.0	5558.97	3913.52	56981.08	1.0	0.0	-1	1.0	0	...	1.0	0	3457
7	0.0	1.0	-15208.03	-1783.93	265210.55	1.0	0.0	1	1.0	0	...	1.0	0	-4478
8	0.0	1.0	-109.04	8239.25	191486.94	1.0	0.0	-1	1.0	0	...	1.0	0	-2083
9	0.0	1.0	15175.26	93142.09	379269.30	1.0	0.0	1	1.0	0	...	1.0	0	3295

10 rows x 26 columns

```
In [ ]: |
```

Future steps

What is next?

Future steps

What is next?

Hadoop-XrootD Connector:

- Add GRID Authentication alongside with Kerberos
- Introduce optimizations and tuning to decrease the performance gap between EOS and HDFS

Future steps

What is next?

Hadoop-XrootD Connector:

- Add GRID Authentication alongside with Kerberos
- Introduce optimizations and tuning to decrease the performance gap between EOS and HDFS

spark-root:

- Implement the ability to write to ROOT files
- Add IO statistics support

Future steps

What is next?

Hadoop-XrootD Connector:

- Add GRID Authentication alongside with Kerberos
- Introduce optimizations and tuning to decrease the performance gap between EOS and HDFS

spark-root:

- Implement the ability to write to ROOT files
- Add IO statistics support

Investigate scaling behavior for larger input (goal is 1 PB)

Future steps

What is next?

Hadoop-XrootD Connector:

- Add GRID Authentication alongside with Kerberos
- Introduce optimizations and tuning to decrease the performance gap between EOS and HDFS

spark-root:

- Implement the ability to write to ROOT files
- Add IO statistics support

Investigate scaling behavior for larger input (goal is 1 PB)

Investigate the possibility to use Spark over Openstack

Summary and Outlook

Summary and Outlook

We have solved two important challenges:

- Hadoop-XrootD Connector can directly access files from the EOS Service
- Spark-root can read ROOT files and infer their schema into Spark

Summary and Outlook

We have solved two important challenges:

- Hadoop-XrootD Connector can directly access files from the EOS Service
- Spark-root can read ROOT files and infer their schema into Spark

Our long term goal is to provide a unified Spark service:

- accessible via Notebook Software
- over Hadoop Clusters or over Openstack with Kubernetes or Apache Mesos

Summary and Outlook

We have solved two important challenges:

- Hadoop-XrootD Connector can directly access files from the EOS Service
- Spark-root can read ROOT files and infer their schema into Spark

Our long term goal is to provide a unified Spark service:

- accessible via Notebook Software
- over Hadoop Clusters or over Openstack with Kubernetes or Apache Mesos

This will simplify the existing physics data analysis procedures by:

- improving resource utilization
- providing user-friendly environments
- reducing the overall “time-to-physics”



Questions?

emotes@cern.ch