

Big Data Tools and Pipelines for Machine Learning in HEP

CERN-EP/IT Data Science Seminar

December 4th, 2019

Luca Canali, Hadoop and Spark Service, IT-DB, CERN

Why “Big Data” Ecosystem for HEP?

- Platforms, tools and R&D
 - Large amounts of **innovation** by open source communities, industry, academia
 - Address key challenges for **data intensive** domains
 - **Lower cost** of development and licensing
- Use of mainstream technologies (Data, ML/AI)
 - Create opportunities for **collaboration**
 - With other **sciences** (astronomy, biology, etc) + with **industry**
 - Talent flow: job market for data scientists and data engineers

Data Engineering to Enable Effective ML

- From “Hidden Technical Debt in Machine Learning Systems”, D. Sculley et al. (Google), paper at NIPS 2015

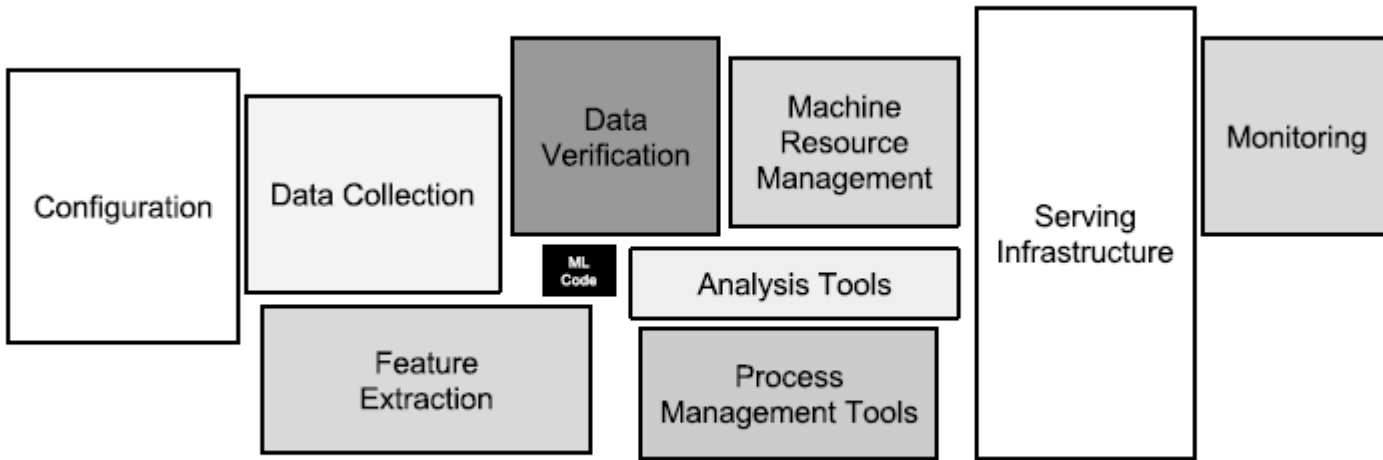
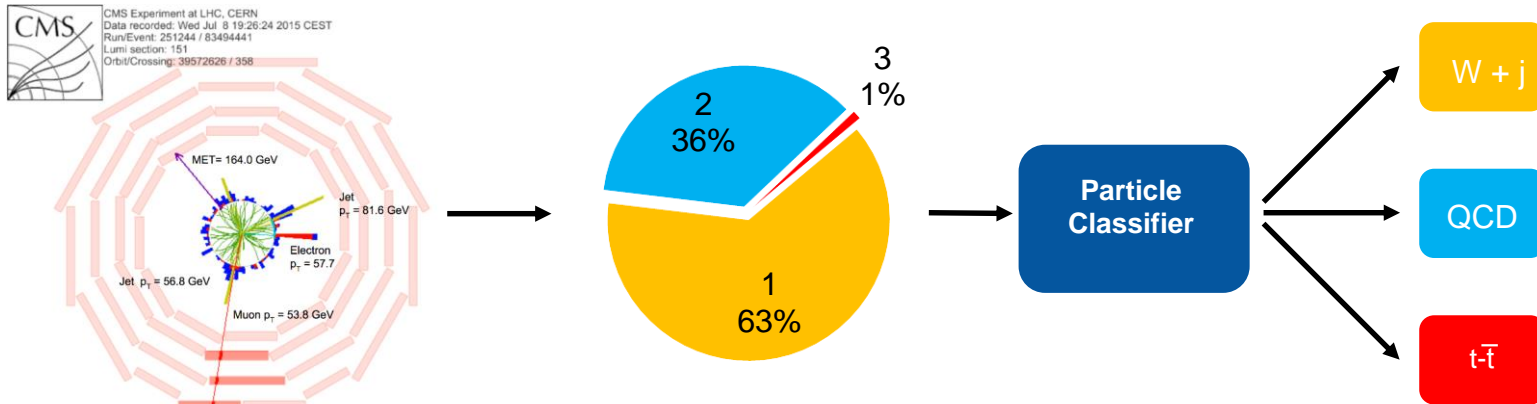


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Use Case: End-to-End ML Pipeline

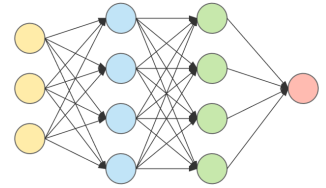
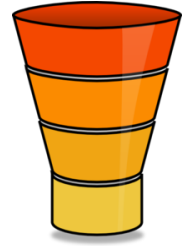
Particles Classifier Using Neural Networks

- R&D to improve the **quality of filtering systems**
 - **Develop** a “Deep Learning classifier” to be used by the filtering system
 - **Goal:** Identify events of **interest** for physics and reduce false positives
 - False positives have a **cost**, as wasted storage bandwidth and computing
 - “Topology classification with deep learning to improve real-time event selection at the LHC”, Nguyen et al. **Comput.Softw.Big Sci. 3 (2019) no.1, 12**

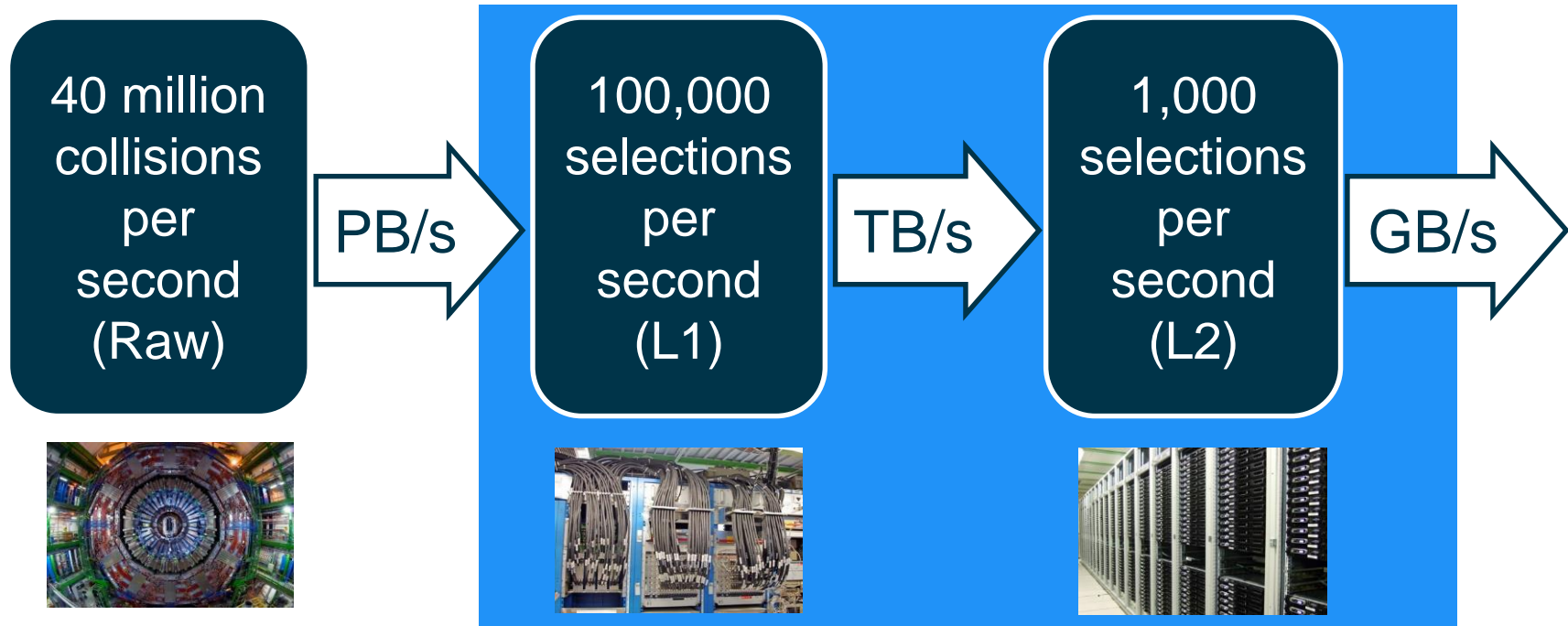


R&D – Data Pipelines

- Improve the **quality of filtering systems**
 - Reduce false positive rate
 - Complement or replace rule-based algorithms with classifiers based on Deep Learning
- Advanced analytics **at the edge**
 - Avoid wasting resources in offline computing
 - Reduction of operational costs

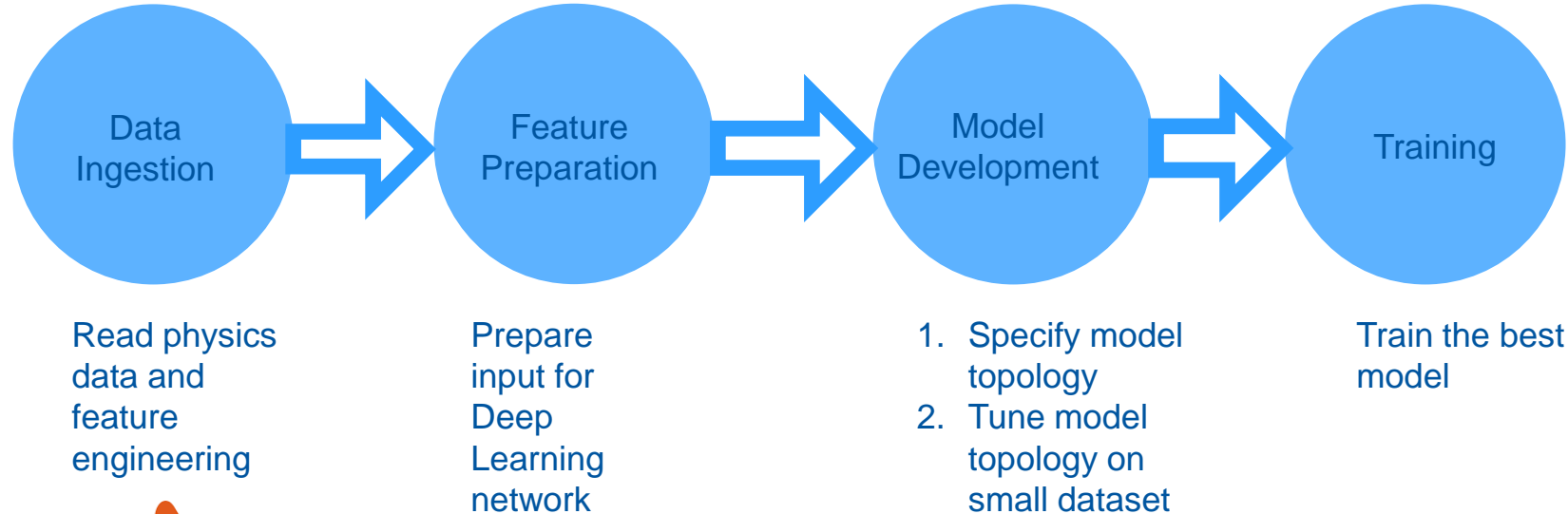


Data Flow at LHC Experiments



This can generate up to a **petabyte** of raw data per second
Reduced to **GB/s** by filtering in real time
Key is how to select potentially interesting events (**trigger** systems).

Deep Learning Pipeline for Physics Data



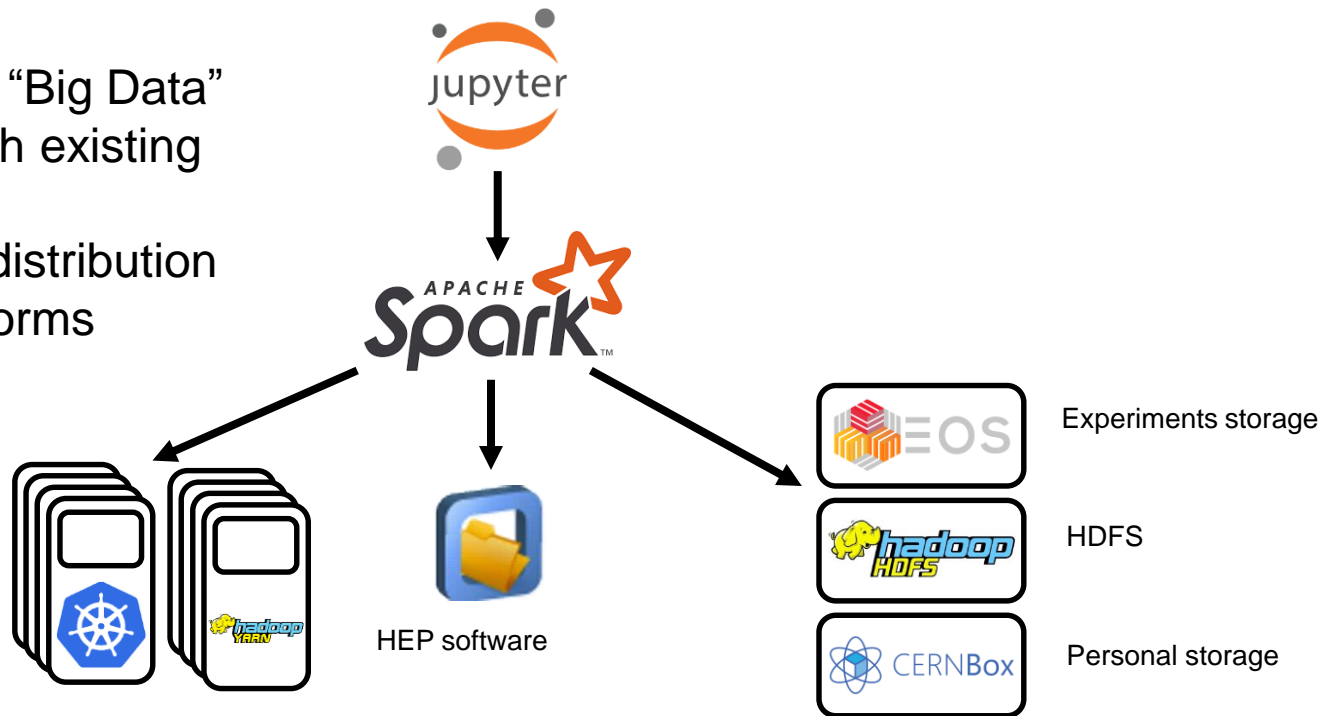
Technology: the pipeline uses Apache Spark + Analytics Zoo and TensorFlow/Keras. Code on Python Notebooks.



CERN SWAN with Apache Spark, a Data Analysis Platform at Scale

Integrating new “Big Data” components with existing infrastructure:

- **Software** distribution
- **Data** platforms



Do the heavylifting in spark and collect aggregated view to panda DF

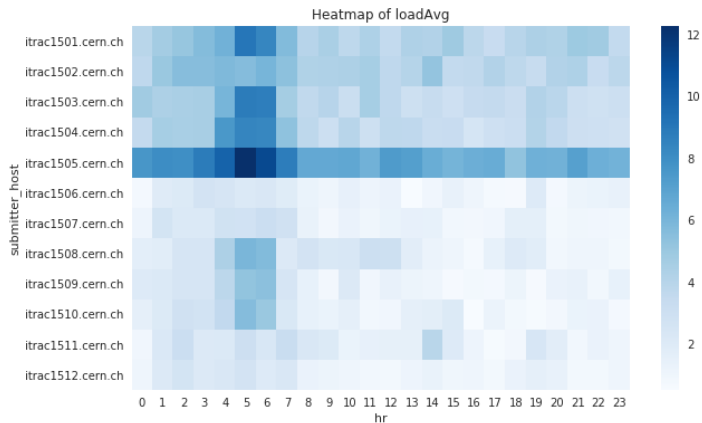
```
In [11]: df_loadAvg_pandas = spark.sql("SELECT submitter_host, \
    avg(body.LoadAvg) as avg, \
    hour(from_unixtime(timestamp / 1000, 'yyyy-MM-dd HH:mm:ss')) as hr \
    FROM loadAvg \
    WHERE submitter_hostgroup = 'hadoop/itdb/datanode' \
    AND dayofmonth(from_unixtime(timestamp / 1000, 'yyyy-MM-dd HH:mm:ss')) = 15 \
    GROUP BY hour(from_unixtime(timestamp / 1000, 'yyyy-MM-dd HH:mm:ss'), submitter_host")\
    .toPandas()
```

Job ID	Job Name	Status	Stages	Tasks	Submission Time	Duration
3	toPandas	COMPLETED	2/2	388 / 388	4 minutes ago	36s

Visualize with seaborn

```
In [19]: # heatmap of service availability
plt.figure(figsize=(10, 6))
ax = sns.heatmap(df_loadAvg_pandas.pivot(index='submitter_host', columns='hr', values='avg'), cmap="Blues")
ax.set_title("Heatmap of loadAvg")
```

Out[19]: Text(0.5,1,u'Heatmap of loadAvg')



Text

Code

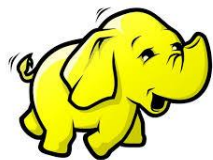
Monitoring

Visualizations



Spark Clusters at CERN: on Hadoop and on Cloud

- Clusters run on
 - Hadoop clusters: Spark on **YARN**
 - Cloud: Spark on **Kubernetes**
- Hardware: commodity servers, continuous refresh and capacity expansion

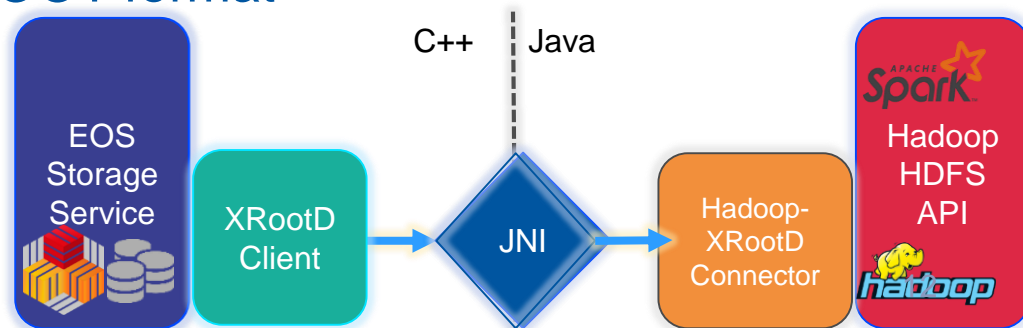


NXCals for Accelerator Logging (part of LHC infrastructure)	Hadoop - YARN – 32 nodes (Cores - 1024, Mem - 16 TB, Storage – 7.5 PB)
General Purpose (Analytix + Hadalytic)	Hadoop - YARN, 54 nodes (Cores – 1184, Mem – 21 TB, Storage – 11 PB)
Cloud containers	Kubernetes on Openstack VMs, Cores - 250, Mem – 2 TB Storage: remote HDFS or EOS (for physics data)

Extending Spark to Read Physics Data

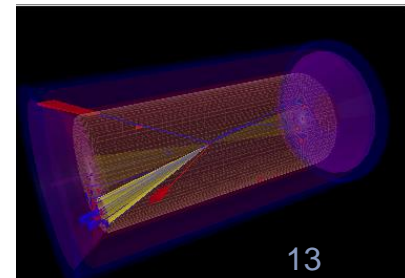
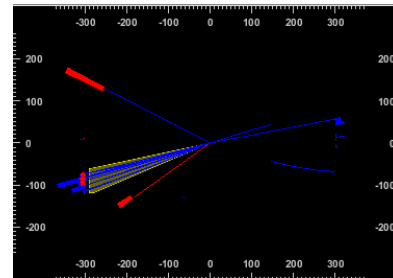
- Physics data
 - Currently: >300 PBs of Physics data, increasing ~90 PB/year
 - Stored in the CERN EOS storage system in ROOT Format and accessible via XRootD protocol
- Integration with Spark ecosystem
 - Hadoop-XRootD connector, HDFS compatible filesystem
 - Spark Datasource for ROOT format

<https://github.com/cerndb/hadoop-xrootd>
<https://github.com/diana-hep/spark-root>



Labeled Data for Training and Test

- Simulated events
 - Software **simulators** are used to generate events and calculate the detector response
 - **Raw data** contains **arrays** of simulated particles and their properties, stored in **ROOT** format
 - 54 million events

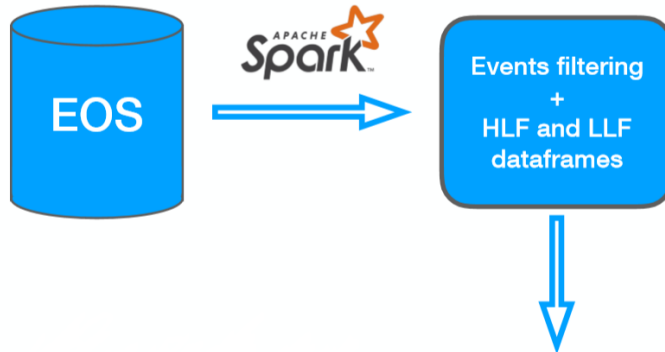


Step 1: Data Ingestion

- **Read** input files: **4.5 TB** from ROOT format
- Feature engineering
 - Python and **PySpark** code, using Jupyter notebooks
- Write output in Parquet format

Input:

- 54 M events
~4.5 TB
- Physics data storage (EOS)
- Physics data format (ROOT)



Output:

- 25 M events
- 950 GB in Parquet format
- Target storage (HDFS or EOS)¹⁴

Spark DataFrames – Some Basics

- Data in Apache Spark
 - The key abstraction and API is **DataFrame**
 - Think of it as “a distributed version of **Pandas** DF”
- Can parallelize/distribute I/O and operations
 - Large choice of **data formats** for input and output (extendable)
 - Can do I/O with **HDFS, EOS, S3**, local filesystem, ...
 - Scale out: actions operate in parallel with data partition granularity, run on cluster resources of choice (YARN, K8S, local machine)

```
myDF = spark.read.format("root").load("root://eos...")  
myDF.count()
```

Feature Engineering

- Filtering
 - Multiple filters, keep only events of interest
 - Example: “events with one electrons or muon with Pt > 23 Gev”
- Prepare “Low Level Features”
 - Every event is associated to a matrix of particles and features (801x19)

```
features = [  
    'Energy', 'Px', 'Py', 'Pz', 'Pt', 'Eta', 'Phi',  
    'vtxX', 'vtxY', 'vtxZ', 'ChPFIso', 'GammaPFIso', 'NeuPFIso',  
    'isChHad', 'isNeuHad', 'isGamma', 'isEle', 'isMu', 'Charge'  
]
```

- High Level Features (HLF)
 - **Additional 14 features** are computed from low level particle features
 - Calculated based on domain-specific knowledge using Python code

Step 2: Feature Preparation

Features are converted to formats suitable for training

- **One Hot Encoding** of categories
- MinMax scaler for High Level Features
- **Sorting** Low Level Features: prepare input for the sequence classifier, using a metric based on physics. This use a Python **UDF**.
- Undersampling: use the same number of events for each of the three categories

Result

- 3.6 Million events, **317 GB**
- Shuffled and split into training and test datasets
- Code: in a Jupyter notebook using **PySpark** with Spark SQL and ML

Feature preparation

Elements of the hfeatures column are list, hence we need to convert them into Vectors.Dense

```
In [10]: from pyspark.ml.linalg import Vectors, VectorUDT
         from pyspark.sql.functions import udf

         vector_dense_udf = udf(lambda r : Vectors.dense(r),VectorUDT())
         data = data.withColumn('hfeatures_dense',vector_dense_udf('hfeatures'))
```

Now we can build the pipeline to scale HLF and encode the labels

```
In [11]: from pyspark.ml import Pipeline
         from pyspark.ml.feature import OneHotEncoderEstimator
         from pyspark.ml.feature import MinMaxScaler

         ## One-Hot-Encode
         encoder = OneHotEncoderEstimator(inputCols=["label"],
                                         outputCols=["encoded_label"],
                                         dropLast=False)

         ## Scale feature vector
         scaler = MinMaxScaler(inputCol="hfeatures_dense",
                               outputCol="HLF_input")

         pipeline = Pipeline(stages=[encoder, scaler])


         %time fitted_pipeline = pipeline.fit(data)

         CPU times: user 294 ms, sys: 293 ms, total: 587 ms
         Wall time: 1min 34s

In [12]: data = fitted_pipeline.transform(data)
```

Performance - Lessons Learned

- Data preparation is **CPU bound**
 - Heavy serialization-deserialization due to Python UDF
- Ran using 400 cores: data ingestion took ~3 hours
- It can be **optimized**, but is it worth it ?
 - Use Spark SQL, or Scala instead of Python UDF
 - Optimization: replaced parts of Python UDF code with Spark SQL and **higher order functions**: run time, from 3 hours to **2 hours**



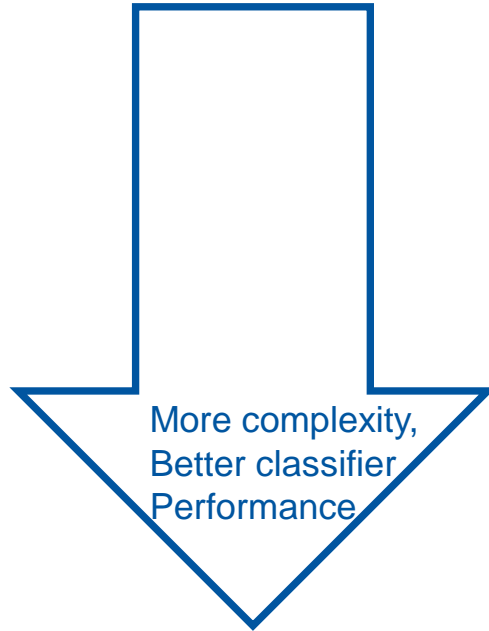
```
FILTER(Electron,  
      electron -> electron.PT > 23  
) Electron,  
FILTER(MuonTight,  
      muon -> muon.PT > 23  
) MuonTight
```

```
WHERE cardinality(Electron) > 0  
      OR cardinality(MuonTight) > 0
```

Development Practices

- **Development**, start small
 - Use a subset of data for development
 - Use SWAN or a local laptop/desktop
- Run at **scale** on clusters
 - Same code runs at scale on clusters: YARN, K8S
 - Smooth transition, need for some additional config (e.g. memory)
- API and code lifecycle
 - Spark DataFrame API is stable and popular
 - Improve **collaboration** with different teams, reproducibility, maintainability

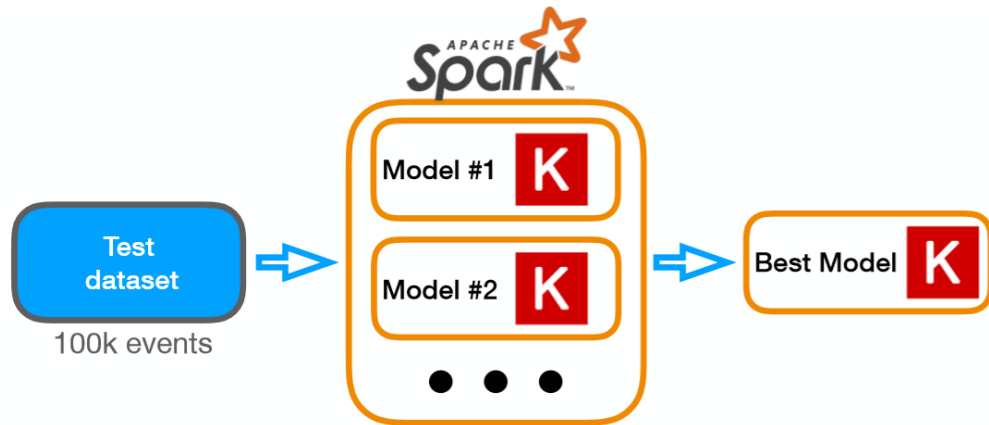
Neural Network Models



1. Fully connected **feed-forward** deep neural network
 - Trained using High Level Features (~1 GB of data)
2. Neural network based on Gated Recurrent Unit (**GRU**)
 - Trained using Low Level Features (~ 300 GB of data)
3. Inclusive classifier model
 - Combination of (1) + (2)

Hyper-Parameter Tuning– DNN

- Hyper-parameter tuning of the DNN model
 - Trained with a subset of the data (cached in memory)
 - **Parallelized with** Spark, using `spark_sklern.grid_search`
 - And scikit-learn + keras: `tensorflow.keras.wrappers.scikit_learn`

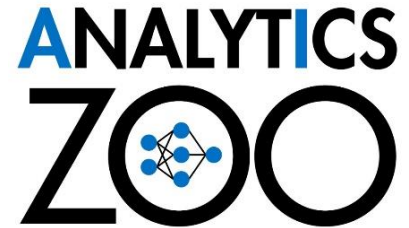


Deep Learning at Scale with Spark

- Investigations and constraints for our exercise
- How to run deep learning in a Spark data pipeline?
 - Neural network models written using **Keras API**
 - Deploy on **Hadoop** and/or **Kubernetes** clusters (CPU clusters)
- **Distributed** deep learning
 - GRU-based model is complex
 - **Slow** to train on a single commodity (CPU) server

Spark, Analytics Zoo and BigDL

- **Apache Spark**
 - Leading tool and API for data processing at scale
- **Analytics Zoo** is a platform for **unified** analytics and AI
 - Runs on Apache Spark leveraging BigDL / Tensorflow
 - For service developers: integration with infrastructure (hardware, data access, operations)
 - For users: Keras APIs to run user models, integration with Spark data structures and pipelines
- **BigDL** is an open source distributed deep learning framework for Apache Spark



BigDL Runs as Standard Spark Programs

Standard Spark jobs

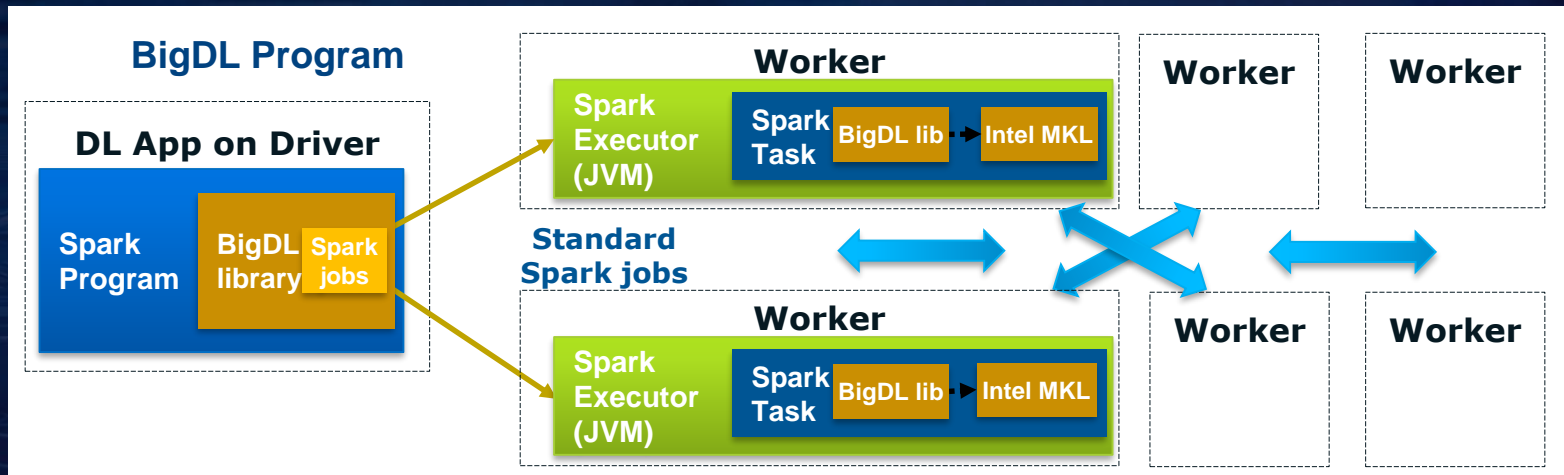
- No changes to the Spark or Hadoop clusters needed

Iterative

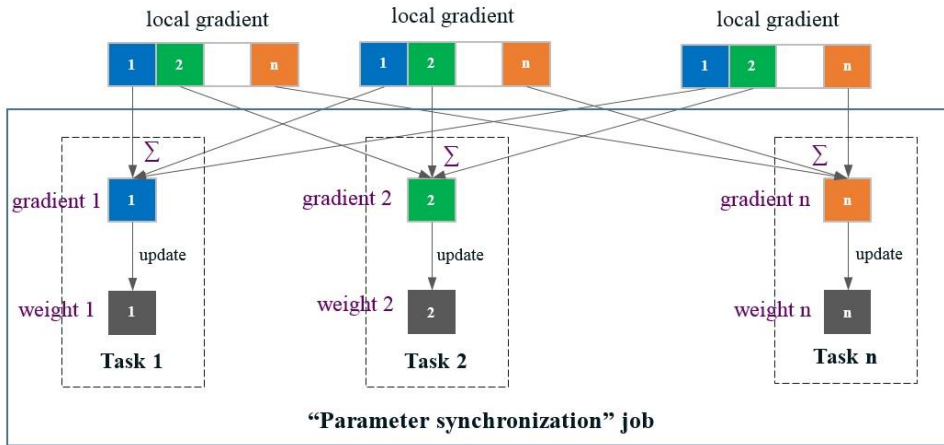
- Each iteration of the training runs as a Spark job

Data parallel

- Each Spark task runs the same model on a subset of the data (batch)



BigDL Parameter Synchronization



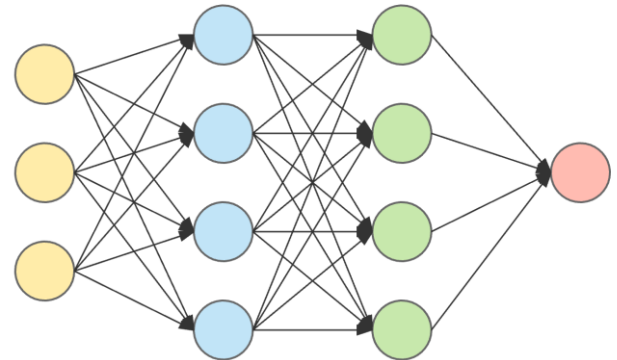
```
For each task n in the "parameter synchronization" job
  shuffle the nth partition of all gradients to this task
  aggregate (sum) the gradients
  updates the nth partition of the weights
  broadcast the nth partition of the updated weights
}
```

Source: <https://github.com/intel-analytics/BigDL/blob/master/docs/docs/whitepaper.md>

Model Development – DNN for HLF

- Model is instantiated using the Keras-compatible API provided by Analytics Zoo

```
In [7]: # Create keras like zoo model.  
# Only need to change package name from keras to zoo.pipeline.api.keras  
  
from zoo.pipeline.api.keras.optimizers import Adam  
from zoo.pipeline.api.keras.models import Sequential  
from zoo.pipeline.api.keras.layers.core import Dense, Activation  
  
model = Sequential()  
model.add(Dense(50, input_shape=(14,), activation='relu'))  
model.add(Dense(20, activation='relu'))  
model.add(Dense(10, activation='relu'))  
model.add(Dense(3, activation='softmax'))  
  
creating: createZooKerasSequential  
creating: createZooKerasDense  
creating: createZooKerasDense  
creating: createZooKerasDense  
creating: createZooKerasDense
```



Model Development – GRU + HLF

A more complex network topology, combining a GRU of Low Level Feature + a DNN of High Level Features

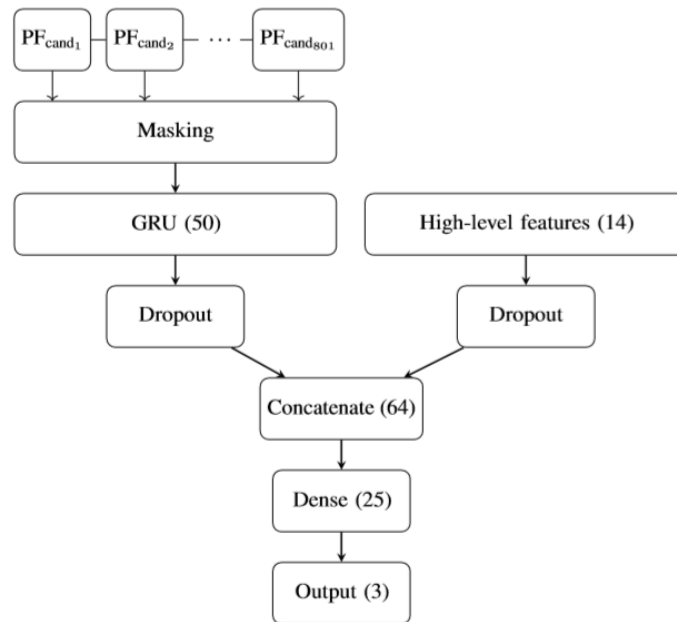
```
from zoo.pipeline.api.keras.optimizers import Adam
from zoo.pipeline.api.keras.models import Sequential
from zoo.pipeline.api.keras.layers.core import *
from zoo.pipeline.api.keras.layers.recurrent import GRU
from zoo.pipeline.api.keras.engine.topology import Merge

## GRU branch
gruBranch = Sequential() \
    .add(Masking(0.0, input_shape=(801, 19))) \
    .add(GRU(
        output_dim=50,
        activation='tanh'
    )) \
    .add(Dropout(0.2)) \

## HLF branch
hlfBranch = Sequential() \
    .add(Dropout(0.2, input_shape=(14,)))

## Concatenate the branches
branches = Merge(layers=[gruBranch, hlfBranch], mode='concat')

## Create the model
model = Sequential() \
    .add(branches) \
    .add(Dense(25, activation='relu')) \
    .add(Dense(3, activation='softmax'))
```



Distributed Training

Instantiate the estimator using Analytics Zoo / BigDL

```
# Create SparkML compatible estimator for deep Learning training

from bigdl.optim.optimizer import EveryEpoch, Loss, TrainSummary, ValidationSummary
from zoo.pipeline.nnframes import *
from zoo.pipeline.api.keras.objectives import CategoricalCrossEntropy

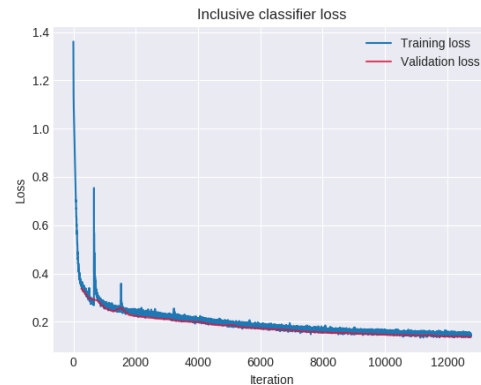
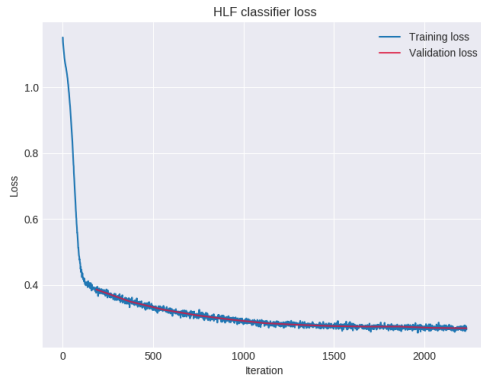
estimator = NNEstimator(model, CategoricalCrossEntropy()) \
    .setOptimMethod(Adam()) \
    .setBatchSize(BDLbatch) \
    .setMaxEpoch(numEpochs) \
    .setFeaturesCol("HLF_input") \
    .setLabelCol("encoded_label") \
    .setValidation(trigger=EveryEpoch(), val_df=testDF,
                  val_method=[Loss(CategoricalCrossEntropy())], batch_size=BDLbatch)
```

The actual training is distributed to Spark executors

```
%%time
trained_model = estimator.fit(trainDF)
```

Storing the model for later use

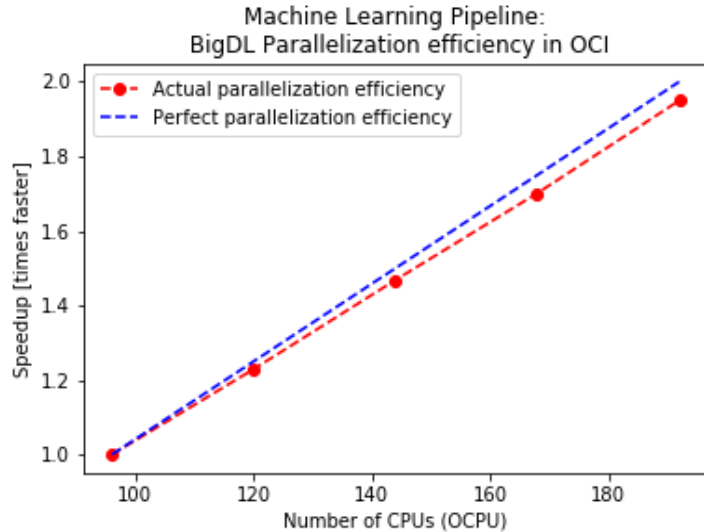
```
modelDir = logDir + '/nnmodels/HLFClassifier'
trained_model.save(modelDir)
```



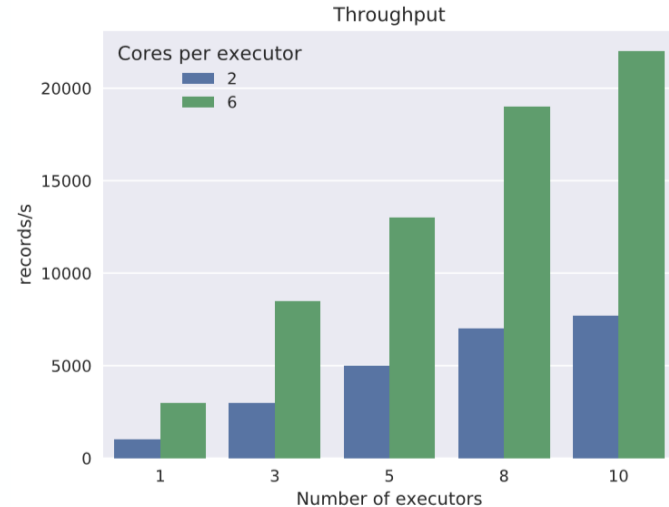
Performance and Scalability of Analytics Zoo/BigDL

Analytics Zoo/BigDL on Spark scales up in the ranges tested

Inclusive classifier model

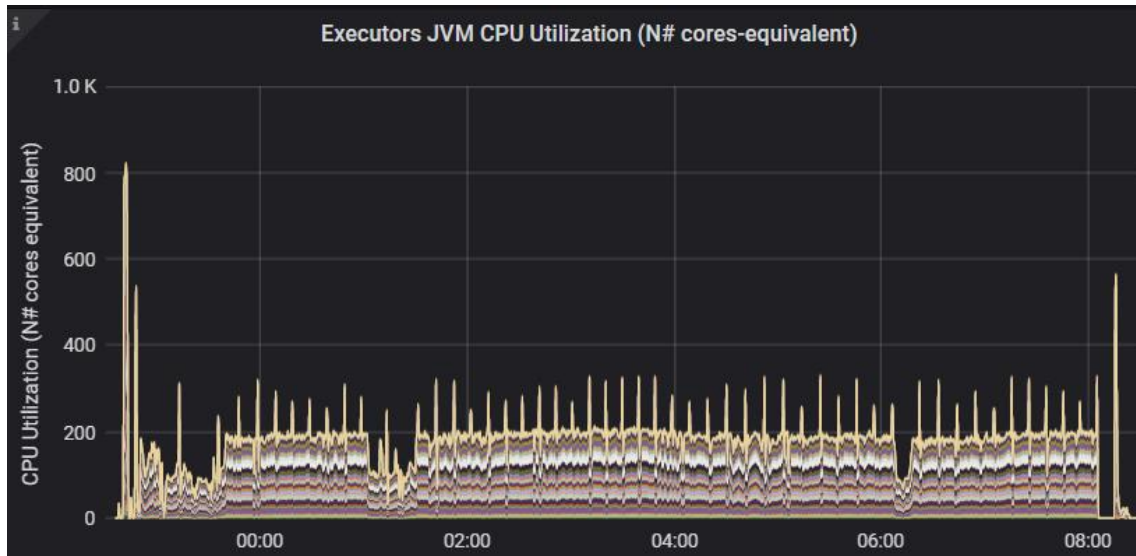


DNN model, HLF features



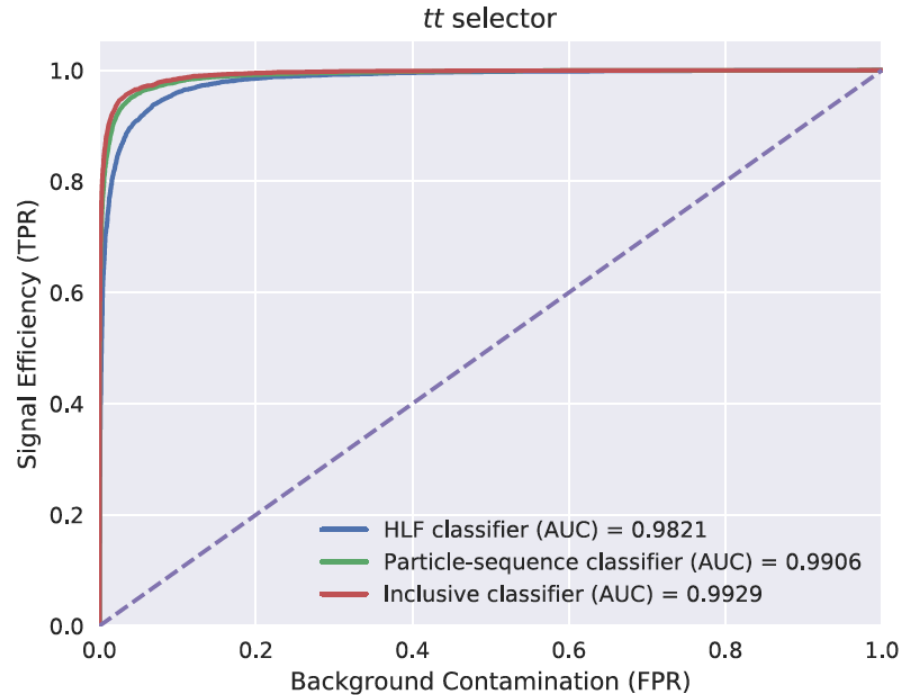
Workload Characterization

- Training with Analytics zoo
 - GRU-based model: Distributed training on YARN cluster
 - Measure with Spark Dashboard: it is **CPU bound**





Results – Model Performance

- Trained models with Analytics Zoo and BigDL
- Met the expected results for model performance: ROC curve and AUC



Spark + TensorFlow

- Additional tests on different architecture
 - Data preparation ->  Apache Spark™
 - Exchange data with TFRecord format
 - Distributed DL ->  TensorFlow

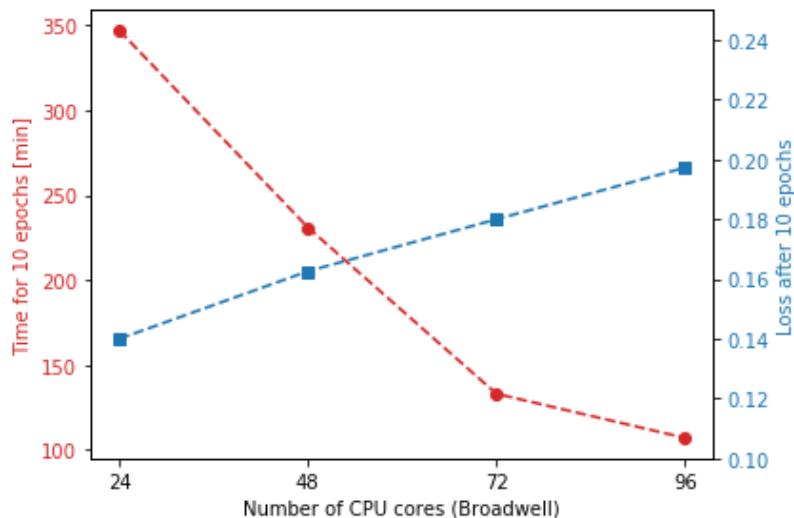
Training with TensorFlow 2.0

- Training and test data
 - Converted from Parquet to **TFRecord** format using **Spark**
 - TensorFlow: data ingestion using **tf.data** and **tf.io**
- Distributed training with **tf.distribute** + tool for **K8S**: <https://github.com/cerndb/tf-spawner>

Distributed training with TensorFlow 2.0 on Kubernetes (CERN cloud)

TF 2.0 feature:

```
tf.distribute.experimental.  
MultiWorkerMirroredStrategy
```

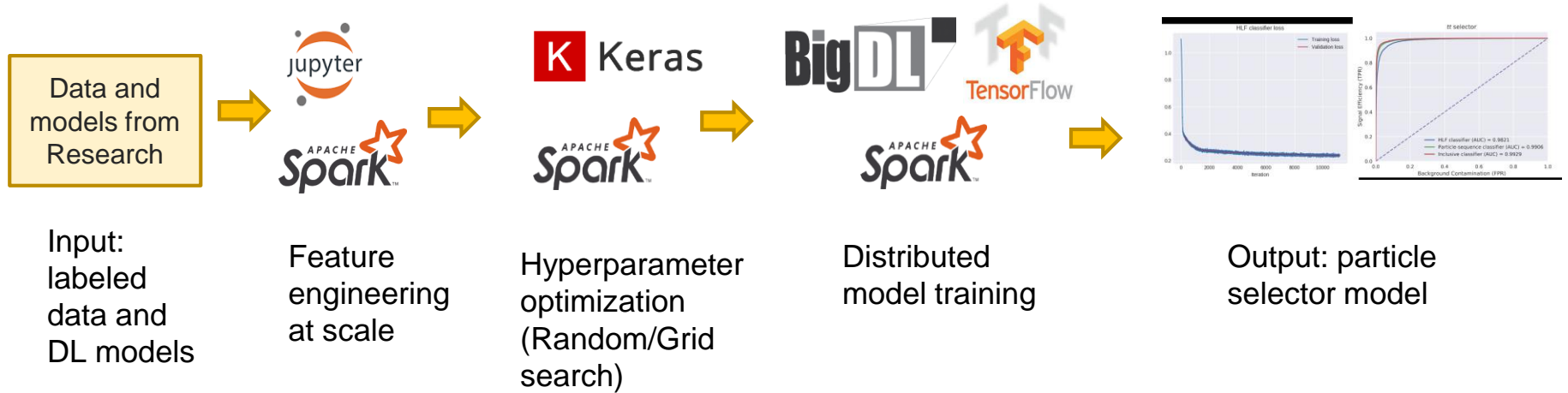


Performance and Lessons Learned

- Measured distributed training elapsed time
 - From a few hours to 11 hours, depending on model, number of **epochs** and **batch size**. Hard to **compare** different methods and solutions (many parameters)
- Distributed training with BigDL and Analytics Zoo
 - Integrates very well with Spark
 - Need to **cache** data in memory
 - Noisy clusters with stragglers can add latency to **parameter synchronization**
- TensorFlow 2.0
 - It is straightforward to **distribute** training on CPUs and GPUs with tf.distribute
 - Data flow: Use TFRecord format, read with TensorFlow's **tf.data** and tf.io
 - GRU training performance on **GPU**: 10x speedup in TF 2.0
 - Training of the Inclusive Classifier on a single P100 in 5 hours

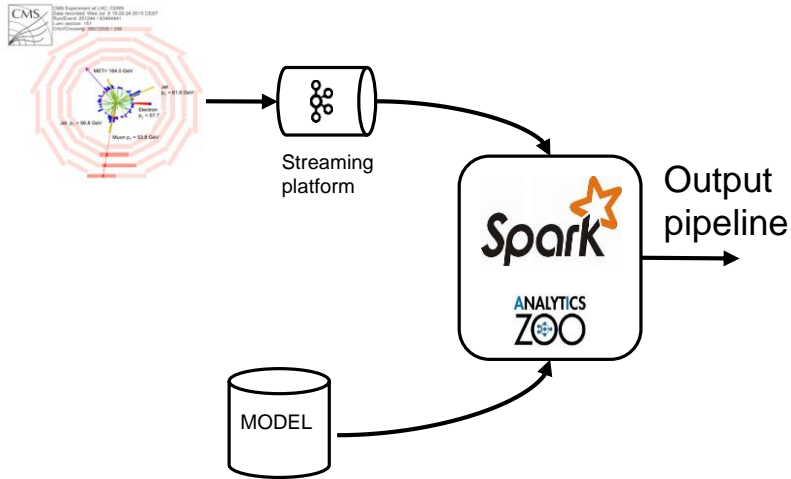


Recap: our Deep Learning Pipeline

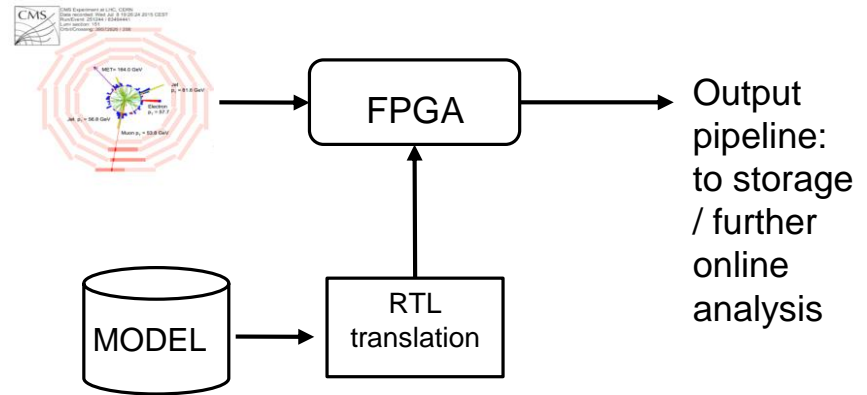


Model Serving and Future Work

- Using Apache Kafka and Spark?



- FPGA serving DNN models



End-To-End ML Pipeline Summary

- Spark, Python notebooks
 - Provide **well-known APIs** and productive environment for data preparation
- Data preparation performance, lessons learned:
 - Use Spark SQL/**DataFrame API**, avoid Python UDF when possible
- Successfully **scaled Deep Learning** on Spark clusters
 - Using Analytics Zoo and BigDL
 - Deployed on existing Intel Xeon-based servers: **Hadoop** clusters and **cloud**
- Good results also with **Tensorflow 2.0**, running on Kubernetes
 - GPU resources are important for DL
- We have only explored some of the available solutions
 - **Data** preparation and **scalable** + distributed **training** are key



Services and Resources

Users of Big Data Platforms

- Many use cases at CERN for analytics
 - Data analysis, dashboards, plots, joining and aggregating multiple data, libraries for specialized processing, machine learning, ...
- Communities
 - **Physics:**
 - Analytics on computing data (e.g. studies of popularity, grid jobs, file transfers, etc) (**CMS Spark** project, ATLAS Rucio)
 - Parallel processing of **ROOT** RDataframes with PyRDF for data analysis
 - Development of new ways to process Physics data, e.g.: data reduction and analysis with spark-ROOT, more recently **Coffea** and **Laurelin** by LHC Bigdata project
 - ATLAS EventIndex project
 - **IT:**
 - Analytics on IT monitoring data
 - Computer security
 - **BE:**
 - **NXCALS** – next generation accelerator logging platform
 - BE controls data and analytics

Hadoop and Spark Service at CERN IT

- Setup and run the infrastructure
- Support user community
 - Provide consultancy
 - Doc and training
- Facilitate use
 - Package libraries and configuration
 - Client machines + Docker clients
 - Notebook service integration
 - <https://hadoop.web.cern.ch>
 - <https://hadoop-user-guide.web.cern.ch>



Hadoop service in numbers

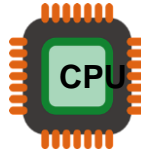


6 clusters

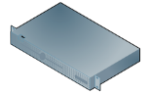
- ❖ 4 production (bare-metal)
- ❖ 2 QA clusters (VMs)



40+ TB of Memory



4000+ physical cores



140+ physical servers

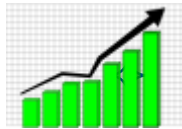


HDDs and SSDs



openstack.

40+ virtual machines



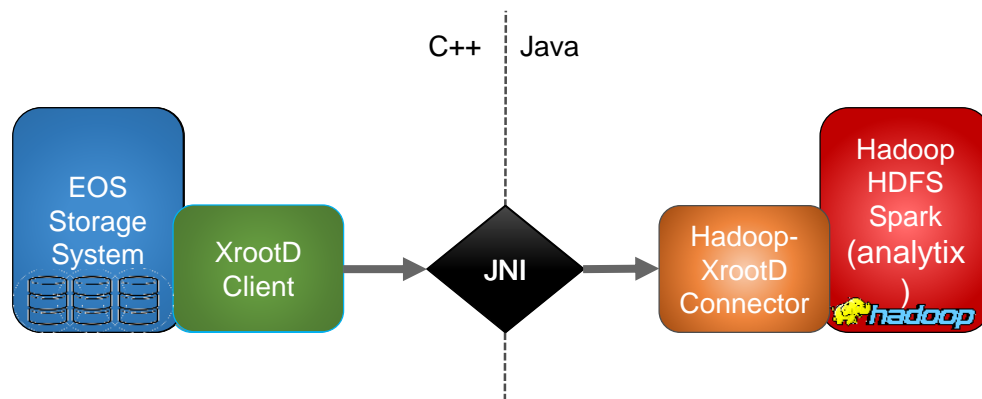
Data growth: ~8 TB per day



28+ PBs of Storage

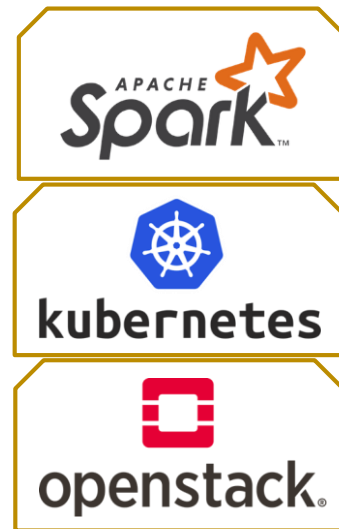
XRootD connector for Hadoop and Spark

- A library that binds Hadoop-based file system API with XRootD native client
 - Developed by CERN IT
- Allows most of components from Hadoop stack (**Spark**, MapReduce, Hive etc) to read/write from **EOS** and **CASTOR** **directly**
 - No need to copy the data to HDFS before processing
 - Works with Grid certificates and Kerberos for authentication



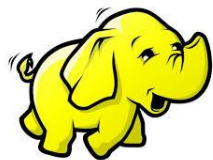
Spark as a service on a private cloud

- Under R&D since 2018, rolled out in 2019
- Appears to be a good solution when data **locality** is not needed
 - CPU and memory intensive rather than IO intensive workloads
 - Reading from storage systems via network (EOS, S3, “foreign” HDFS)
 - Compute resources can be flexibly scaled out
- Spark clusters – on cloud **containers**
 - Kubernetes on Openstack
 - Spark runs on Kubernetes since version 2.3
- Use cases
 - SWAN integration for users reading from EOS
 - High demand of computing resources, needing to use cloud resources
 - Streaming jobs (e.g. accessing Apache Kafka)



Spark Clusters at CERN: on Hadoop and on Cloud

- Clusters run on
 - Hadoop clusters: Spark on **YARN**
 - Cloud: Spark on **Kubernetes**
- Hardware: commodity servers, continuous refresh and capacity expansion



NXCals for Accelerator Logging (part of LHC infrastructure)	Hadoop - YARN – 32 nodes (Cores - 1024, Mem - 16 TB, Storage – 7.5 PB)
General Purpose	Hadoop - YARN, 54 nodes (Cores – 1184, Mem – 21 TB, Storage – 11 PB)
Cloud containers	Kubernetes on Openstack VMs, Cores - 250, Mem – 2 TB Storage: remote HDFS or EOS (for physics data)

SWAN – Jupyter Notebooks On Demand

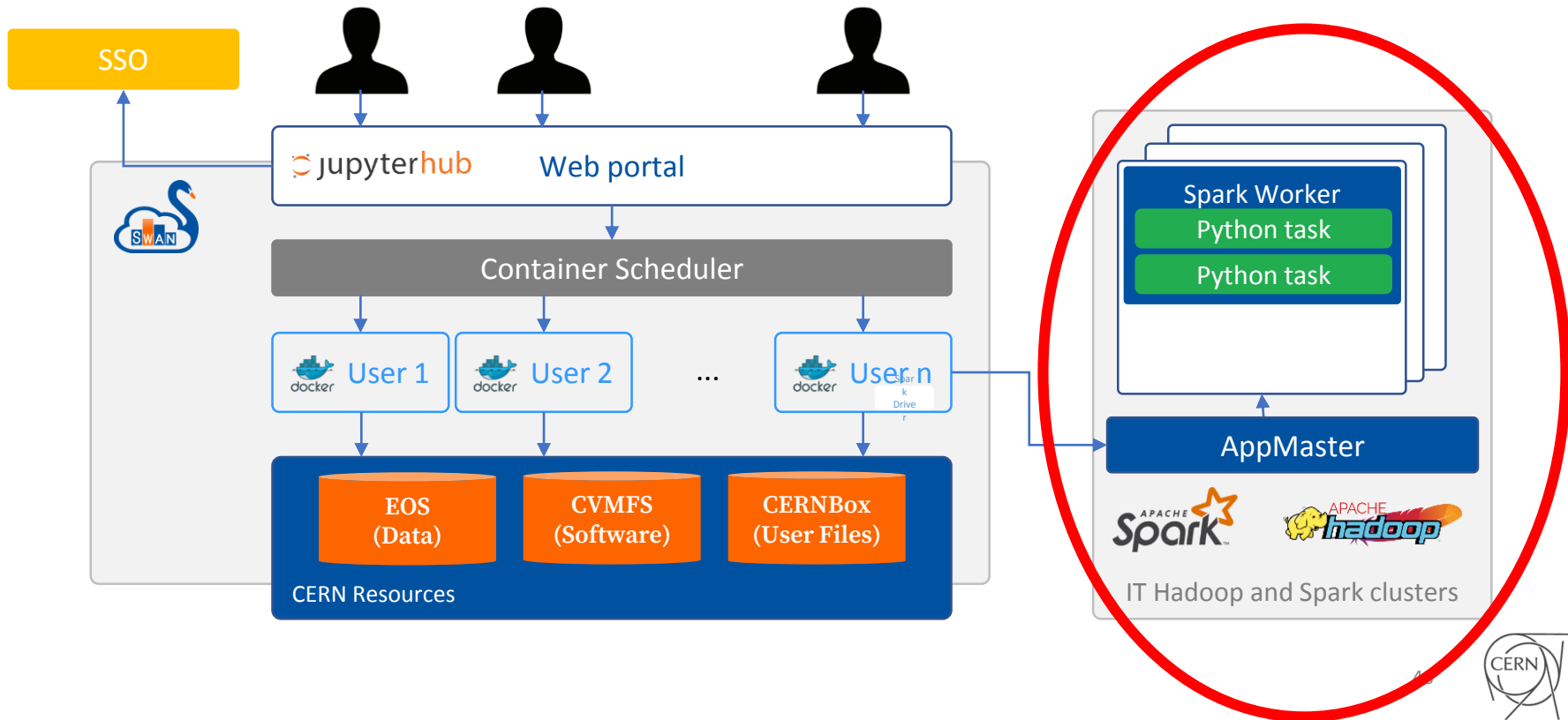


- Service for web based analysis (SWAN)
 - Developed at CERN, initially for physics analysis by EP-SFT
- An interactive platform that combines code, equations, text and visualizations
 - Ideal for exploration, reproducibility, collaboration
- Fully **integrated with Spark and Hadoop** service
 - Python on Spark (PySpark) at scale
 - Modern, powerful and scalable platform for data analysis
 - Web-based: no need to install any software





Spark Integration in the SWAN Architecture



Example Notebooks

<https://swan.web.cern.ch/content/apache-spark>



Apache Spark

SWAN: Spark connector and monitor

These notebooks exemplify the usage of SWAN and Spark for analytics and machine learning use cases at CERN.

Analyzing monitoring data

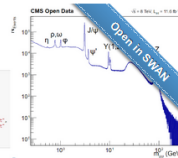
```
Do the heavy lifting in spark and collect aggregated row to pandas DF
df = spark.read.jdbc(url=jdbcUrl, table=jdbcTable, properties=jdbcProperties)
df.toPandas()
```

Analyzing LHC logging data

```
Import data
df = spark.read.jdbc(url=jdbcUrl, table=jdbcTable, properties=jdbcProperties)
df.toPandas()
```

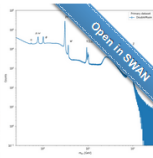
Processing ROOT (NanoAOD) files with Distributed ROOT RDataFrame in Python and Spark (PySpark)

```
Open Spark
spark = SparkSession.builder().appName("ROOT").getOrCreate()
df = spark.read.format("root").option("uri", uri).load()
```



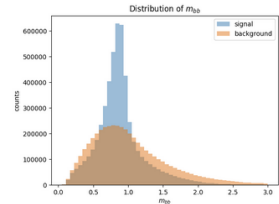
Physics analysis with Apache Spark using Coffea and Laurelin packages

```
from coffea.nanoevents import NanoEventsFactory
factory = NanoEventsFactory.from_config({'root': {'uri': 'root://cmsweb.cern.ch/...'}})
events = factory.from_root(uri)
```



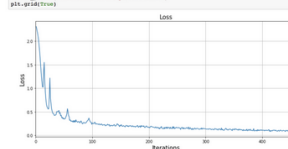
Machine Learning with Apache Spark

```
spark.mllib.trainWithRDD(rdd, classifier)
model = spark.mllib.trainWithRDD(rdd, classifier)
model.saveAsTextFile("output")
```



Handwritten Digit Classification using Apache Spark and BigDL

```
from bigdl.tensorflow import TensorFlow
tf = TensorFlow()
tf.train(train_loader, dev_loader, num_epochs)
```



Why Spark?

- Data processing **at scale**
 - DataFrames, similar to Pandas
 - You cannot “fit the problem on a laptop”
- Machine Learning at scale
 - “scikit-learn” at scale
- Data Streaming
- One tool with many features
 - Popular API



Image Credit: Vector Pocket Knife from Clipart.me

What Can be Improved?

- Spark runs natively on JVM
 - Python integration is key at CERN. Works OK but performance still needs to improve
- Spark is most useful at scale
 - We see many ML tasks that fit in a server
- Spark and GPUs
 - Work in progress. Horovod on Spark is a possibility
- Competition
 - Cloud and open source tools: Kubeflow, DASK, ...

Areas for Future Development

- Further improve the analytics platform
 - Use of cloud resources, also testing public clouds
 - Integration of GPUs
- Machine Learning
 - Collect feedback from users communities
 - Are tools at scale useful?
 - What are the main use cases?
 - What is missing in the platform?

Conclusions

- End-to-end **pipeline** for machine learning
 - Developed with Apache **Spark**, BigDL and Tensorflow, using Jupyter/Python,
- Big Data tools and **platforms** at CERN
 - For data analysis, **machine learning** and streaming
 - Run at scale on YARN and on Kubernetes
 - **Integrate** with CERN computing environment

Acknowledgments and Links

- Matteo Migliorini, Marco Zanetti, Riccardo Castellotti, Michał Bień, Viktor Khristenko, Maria Girone, CERN openlab, CERN Spark and Hadoop service
- Authors of “Topology classification with deep learning to improve real-time event selection at the LHC”, notably Thong Nguyen, Maurizio Pierini
- Intel team for BigDL and Analytics Zoo: Jiao (Jennie) Wang, Sajan Govindan

ML Pipeline:

- Data and code: <https://github.com/cerndb/SparkDLTrigger>
- Machine Learning Pipelines with Modern Big Data Tools for High Energy Physics <http://arxiv.org/abs/1909.10389>

CERN Spark and Hadoop Service:

- <https://hadoop-user-guide.web.cern.ch/hadoop-user-guide/getstart/access.html>
- Spark on SWAN: <https://swan.web.cern.ch/content/apache-spark>

