SWAN Use Cases

Diogo C., Prasanth K., Enric T.
On behalf of the SWAN team

https://cern.ch/swan

Feb 3rd, 2021
CERN Academic Training
Physics Analysis
SWAN can be used for interactive physics analysis
- Usually medium/small size analysis (input data ~ a few GBs)
- Last steps of analysis (e.g. nanoAOD, ntuples)

All the data you need
- CERNBox: user files
- EOS: project spaces, experiment data

All the software you need
- ROOT
- Python data science libraries: numpy, pandas, matplotlib, seaborn, ...
- Python ML libraries: tensorflow, keras, pytorch, ...
- R kernel
- Plus anything you can find in an LCG release (~ 500 packages)
Dimuon spectrum analysis in SWAN

> Physics analysis is one of the most common use cases of SWAN
  ▪ Refer to lecture 1 for examples

> Analysis producing a dimuon spectrum from CMS Open Data
  ▪ Uses ROOT (RDataFrame), NumPy, pandas and matplotlib
  ▪ Accesses public CMS data on EOS
  ▪ Written in C++ and Python
  ▪ Creates interactive plots in a notebook

Open the SWAN training material
ROOT's RDataFrame: columnar data

- Entries or events or rows
- Columns or "branches"
- Can contain any kind of C++ object

<table>
<thead>
<tr>
<th>pt_x</th>
<th>pt_y</th>
<th>pt_z</th>
<th>theta</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
RDataFrame: quick how-to

1. **Build an RDataFrame** object by specifying your dataset

2. **Apply a series of transformations** to your data
   - filter (e.g. apply some cuts) or
   - define new columns

3. **Apply actions** to the transformed data to produce results (e.g. fill a histogram)
RDataFrame: simple code example

1. Build RDataFrame

```cpp
ROOT::RDataFrame d("t", "f.root");
```n
2. Cut on theta

```cpp
auto h = d.Filter("theta > 0").Histo1D("pt");
```n
3. Fill histogram with pt

```cpp
h->Draw();
```
// d2 is a new data-frame, a transformed version of d
auto d2 = d.Filter("x > 0")
    .Define("z", "x*x + y*y");

// make multiple histograms out of it
auto hz = d2.Histo1D("z");
auto hx = d2.Histo1D("x");
Spark/NXCals
Apache Spark

Apache Spark is a unified computing engine and a set of libraries for parallel data processing on computer clusters.

- **Unified**: provide a unified platform for writing big data applications
  - Data loading, SQL queries, streaming computation and machine learning
- **Computing engine**: limits the scope to computations
  - Only handles loading data from storage systems and performing computations on it
  - Can be used with wide variety of persistent storage systems, including HDFS, EOS, Amazon S3 and Azure Storage
- **Libraries**: In addition to standard libraries, spark supports wide array of external libraries (spark-packages.org)
Spark Applications

> Spark Applications consist of a driver process and a set of executor processes

> **Driver Process**
>  - Maintaining information about the Spark Application
>  - Responding to a user’s program or input
>  - And analysing, distributing, and scheduling work across the executors

> **Executor Process**
>  - Executing code assigned to it by the driver
>  - And reporting the state of the computation on that executor back to the driver node
Architecture of Spark Applications
Spark’s APIs - DataFrame

> Spark has two fundamental sets of APIs: the **low-level “unstructured” APIs**, and the **higher-level structured APIs**

> In this talk we only focus on one such higher-level structured APIs - **DataFrame**

> A DataFrame is the most common Structured API and simply represents a **table of data with rows and columns**.

> The list that defines the columns and the types within those columns is called the **schema**.

> You can think of a DataFrame as a spreadsheet with named columns
Spark’s APIs - DataFrame

Spreadsheet on a single machine

Table or Data Frame partitioned across servers in a data center
To allow every executor to perform work in parallel, Spark breaks up the data into chunks called **partitions**.

A partition is a **collection of rows** that sit on one physical machine in your cluster.

A DataFrame’s partitions represent how the data is **physically distributed** across the cluster of machines during execution.

If you have **one partition**, Spark will have a **parallelism of only one**, even if you have thousands of executors.

If you have **many partitions** but only one executor, Spark will still have a parallelism of only one because there is only one computation resource.
df = spark.range(25)

DF is split into partitions

[1,2,3,4,5]
[6,7,8,9,10]
[11,12,13,14,15]
[16,17,18,19,20]
[21,22,23,24]

Partitions are stored in worker’s memory

> **Partitioned**: DF is partitioned and distributed across worker nodes of the cluster

> **Immutable**: does not change once created, can only be transformed into new DFs

> **Cacheable**: hold all the data in a persistent storage like memory (preferrable) or disk

> **Lazy Evaluation**: Data inside DF is not available or transformed until an action is executed
Two types of operations on DFs:

- **Transformations** – lazy operations that return another DF
- **Actions** – operations that trigger computation and return value
Instructing Spark on how you would like to modify the DF are called transformations.

Example:
```
even_df = df.where("n % 2 = 0")
```

Transformations are the core of how we express logic using spark.

Transformations are **lazy** and only actions will trigger computations.

Two Types of transformations: **narrow** and **wide**
An action instructs Spark to compute a result from a series of transformations.

Example: counting number of records in a dataframe:

```
df.count()
```

There are three kinds of actions:

- Actions to view the data in the console
- Actions to collect data to native objects in respective languages
- Actions to write data to storage systems (HDFS, S3, EOS etc)
Use case – DataCenter Operations

https://swan-gallery.web.cern.ch/notebooks/apache_spark1/analytix-hostmetrics-example.html
Use case – NXCals

https://swan-gallery.web.cern.ch/notebooks/apache_spark1/NXCals-example.html
References

> Further Study on Spark

> Contact IT Hadoop and Spark Service
  - https://cern.service-now.com/service-portal?id=service_element&name=Hadoop-Service

> NXcals Project and API
  - http://nxcals-docs.web.cern.ch/current/
SWAN Use Cases

Thank you

Diogo Castro (diogo.castro@cern.ch)
Enric Tejedor (etejedor@cern.ch)
Prasanth Kothuri (prasanth.kothuri@cern.ch)