Profiling
Distributed execution in ROOT’s DistRDF

Giulio Cрогnαletti, Vincenzo Padulano, Enric Tejedor
Objective: Profile the execution of RDataFrame JIT-ed applications, both

- Locally (i.e. RDataFrame)
  In this context performance suffices thanks to ROOT’s cling profiling feature.

- Distributedly (i.e. DistRDF)
  In this context profiling is less trivial, because performance metrics in each node have to be collected and merged.
  This mechanism is implemented in the DistRDF.Profiling submodule

How is this achieved? What are the requirements?
Build ROOT with debug options:

$ cmake -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O2 -g -fno-omit-frame-pointer" -DCMAKE_BUILD_TYPE="RelWithDebInfo" -DLLVM_BUILD_TYPE="RelWithDebInfo"

Make sure to have permission to collect PMU events of interest in all the machines used. Two possible options:

- Restrict counting to user-space, e.g. use `-e cycles:u` (default option in DistRDF.Profiling)
- Lower perf paranoid level
The variable CLING_PROFILE=1 must be set in all the environments where RDataFrame code runs to allow profiling of JIT-ed code.

All debug information packages for used libraries should be installed e.g. python-debug, glibc-debug, etc

Tools used to produce visualizations (e.g. FlameGraph) must be in PATH in all the environments where perf data is processed.

Some example using RDataFrame locally:

Realistic example
W boson mass analysis
W boson analysis (adding busy loops):

JIT-ing time
Still relevant contribution with small datasets

Event Loop
Code here is clearly visible
DistRDF profiling feature

How does it work?

Client

- activate profiling features
- merge pre-processed data (e.g. merge stacks)
- produce visualization (e.g. flamegraph)

Distributed workers

- collect profiling data (e.g. using perf)
- pre-process it (e.g. collapse equal stacks)
Example: Di-muon analysis

from node 1

merged data

from node 2
The proposed solution is contained in **DistRDF.Profiling** module. It allows to activate the feature using the ClingProfile context manager:

```python
RDataFrame = ROOT.RDF.Experimental.Distributed.Dask.RDataFrame
ClingProfile = ROOT.RDF.Experimental.Distributed.ClingProfile
...

df = RDataFrame("Events", files, npartitions=npartitions,
                  daskclient=client)

with ClingProfile(df, perf_options = {...}):
    df = df.Filter(...)
    ...
    df.GetValue()
```

The event loop must be triggered **inside** the context manager.
The precise requirements to build accurate call stacks using Cling’s profiling feature have been identified
(i.e. build root with -g -fno-omit-frame-pointer, etc)

A mechanism to produce the same analysis for the distributed case have been devised and implemented (PR in review!)
(i.e. DistRDF.Profiling submodule)
Thank you for your attention

Questions?

ROOT
Data Analysis Framework
https://root.cern