Distributed Dask-based national facility at INFN

D. Ciangottini, D. Spiga, T. Tedeschi, M. Tracolli

Outline

- Motivations
- Main objectives
- Challenges
- Design and implementation status
- Recap and plans

Motivations

- **Reducing analysis "time to insight"** (training time for newcomers included)
  - Interactivity and user-friendly/standard UI's
- **Single and easily accessible hub**
  - Reducing complexity and maintenance of multiple and slightly overlapping solutions
- Increasing the system **delivered throughput** (evts/s)

On top of this, we would also like to:

- **Provide a single hub (i.e. at national level)**
  - Well, at least starting "national", but not exclusively
- **Aim to harvest even geographically distributed resources (i.e Tier2)**
  - Raising the bar: even opportunistic resources (i.e. GPU @ HPC ... CINECA)

We build on...

Starting from the outcome of many R&D activities carried on in the past few years, we decided to develop a testbed.

- **Caches and data lake studies** (*ESCAPE*, *IDDLS projects*)
- **Cloud and dynamic resources** (*INFN-Cloud*, *DODAS*)
- **Elastic pool of resources,** even opportunistic (*DODAS*, *CINECA* integration)
- **JWT based federated identity** (*Indigo IAM*)

And most importantly: **everything is just a "Lego brick"**

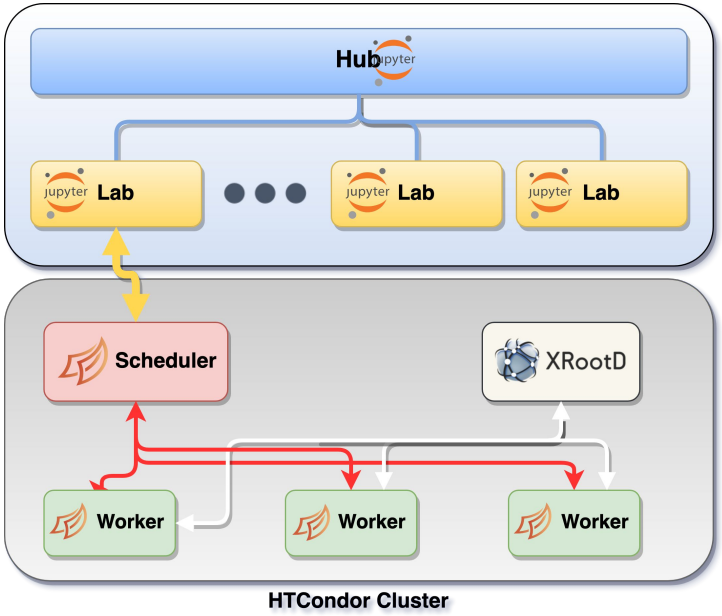
We want to be able to extend/add/remove … and evolve as needed!

Delivering something "standard" and easy to replicate: Ansible & docker & HELM

Current prototype

- **JupyterHub** (JHub) and **JupyterLab** (JLab) to manage the **user-facing part of the infrastructure**
- **Dask** to introduce the **scaling over a batch system**
- **XRootD as data access protocol** toward AAA:
  - Here we foresee the usage of caching layers (see later)

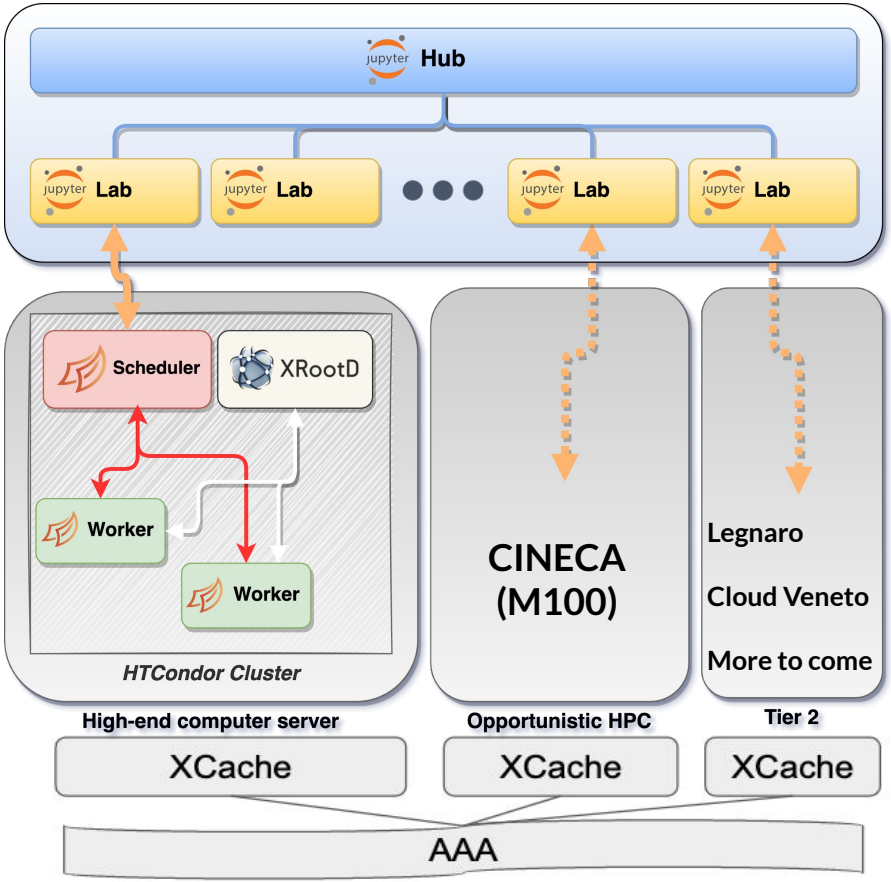
So far, we opted for scaling over HTCondor:
⇒ User prioritization and in general configuration tuning is under study



Geographical extensions

We realized that all this **would be even more powerful if it could be geographically extended** toward any resource provider, including opportunistic resources

⇒ **transparently use T2 resources and exploit HPC** via HTCondor + Dask look like the perfect match so far.



The main challenges

Dask framework for distributed workflow on a batch system has **some clear constraints:**

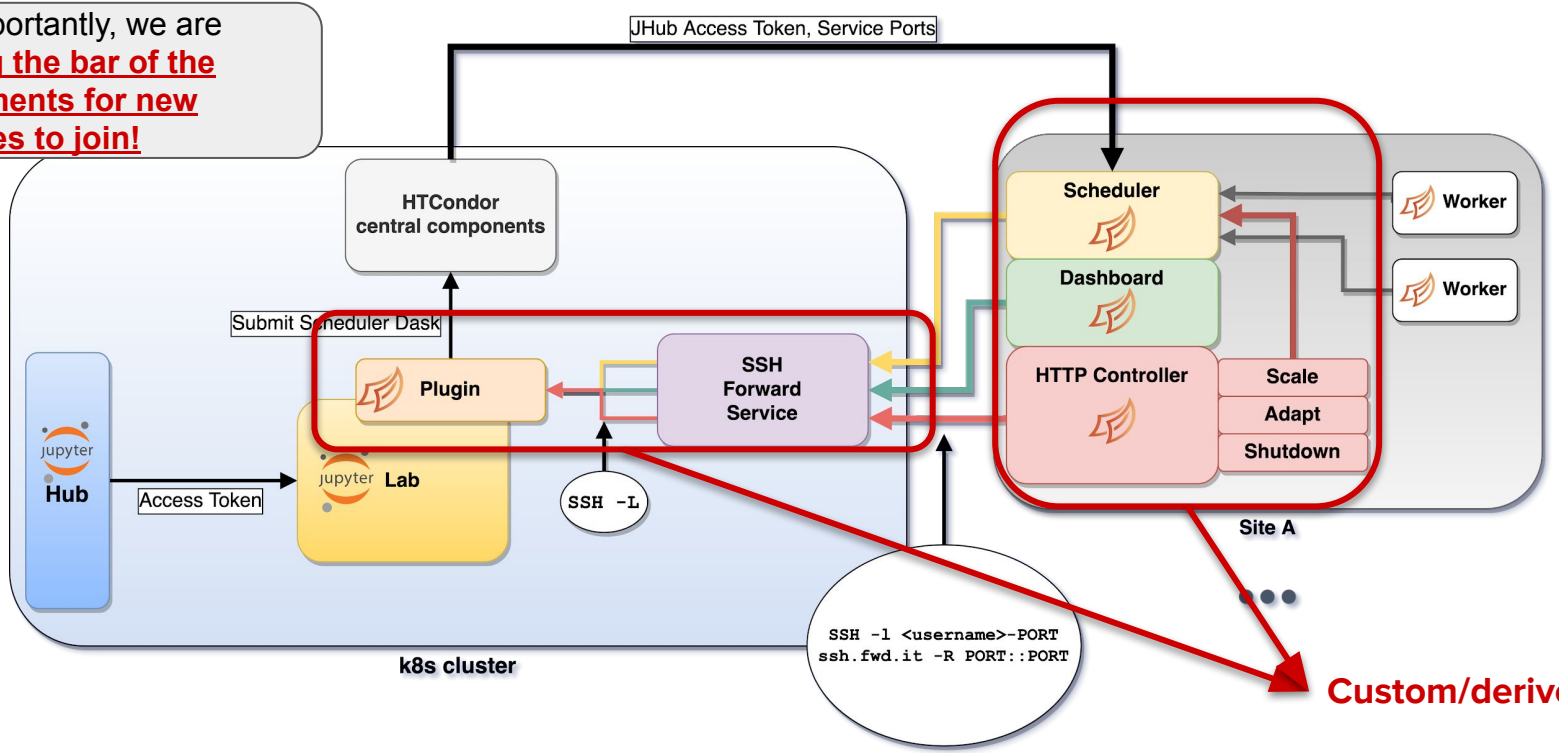
- **Network access** → Dask scheduler should be accessible from the client running on the JupyterLab instance
- **Cluster locality** → Dask scheduler and workers are best suited for running with the same locality (same site/network)

Not only, **data access should also be possible from the remote WNs efficiently** → this is why we consider using xRootd (proxy and/or cache) at different levels

Dask-remote-jobqueue implementation

From the network perspective, **we came up with the following architecture deployed on the current testbed**
This would allow us to respect the **Dask** cluster locality needs, while running code on remote **JupyterLab**

More importantly, we are **lowering the bar of the requirements for new resources to join!**

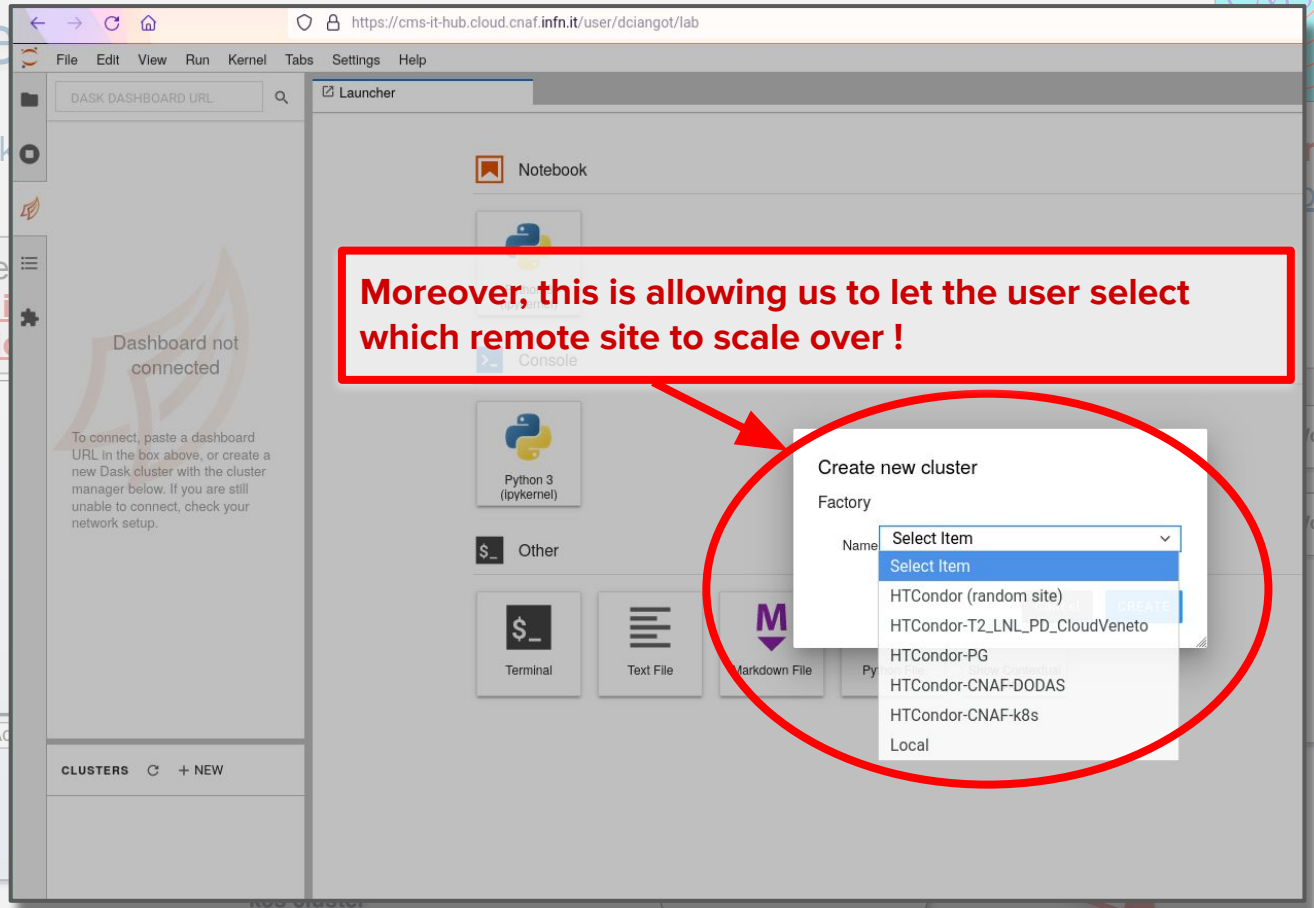


Dask-re

From the network
This would allow

rent testbed
yterLAB

More importantly we
the bar of the requi
new resources to j



Where are we?

We have **developed an e2e testbed system which is now available:**

https://cms-it-hub.cloud.cnaf.infn.it

Please remember that, if interested, you need to be put in the correct user group in order to use the facility!
Drop a message if interested.

Supported workflows:

- **A legacy-friendly batch system** on national resources
  - Accessible from everywhere (lxplus, personal laptop, provided Jupyter instance)
- **An interactive environment based on Jupyter** to develop and run an end-to-end analysis
  - Seamlessly scaling over resources on the batch system
- **A customizable terminal UI** where to run analysis scripts in a semi-interactive scenario
  - Thinking about workflows that can take longer than what the word “interactive” suppose to

Where are we?

- Currently, any data on **AAA can be accessed without the need for a user proxy** in the instance
  - This is happening thanks to a client configuration translating your request into a request **toward the nearest cache system** (both on notebook side and on distributed site node)
- Software on **CVMFS mounted already** on both UI and remote worker nodes
- Still working on **understanding and tuning the system**
- **Ported a VBS analysis in ROOT's RDataFrame to benchmark the system**
  - And also to do a sort of validation of the whole workflow
  - A special thanks go to RDF developers

Our top three priorities now

- **Optimized data serving system** → caches
  - hierarchical layers vs near-site only
  - lazy download vs full streaming
- **Benchmark event throughput and validate** of real analyses with:
  - Different data access patterns
  - Different code bases → Dask task distribution/configuration
- **Scale tests (multiple users, multiple tasks)**
  - Dedicated high-performance machine
  - Scale over T2 site resources
  - Scale over HPC CINECA resources

Summary

- We presented the **overview of the infrastructure testbed we built**
  - **Extensible, Portable, Made by “standards”**
- Integrating the **technical solution in parallel with e2e use case**
  - This is a key, need to improve/increase
- The **data access will be a key now.** “Easy” if everything is local, a bit more challenging if we go beyond locality.
  - We are fully committed to setup/measurement

A final remark: we think building a community here will be crucial, beside the technology it would be also important a common metric definition for benchmarks and comparisons (and thus taking decisions)

Many people contributing at various level

- Diego Ciangottini
- Daniele Spiga
- Mirco Tracolli
- Massimo Biasotto
- Massimo Sgaravatto
- Stefano Nicotri
- Francesco Failla

Tier2-IT:

- M. Biasotto
- G. Donvito
- S. Rahtalou
- E. Mazzoni

Special Thanks to INFN-Cloud

Backup

Technologies

Nothing fancy about the technologies opted in:

- JupyterHub (JHub) and JupyterLab (JLab) to manage the user-facing part of the infrastructure
- Dask to introduce the scaling out toward a batch system
- HTCondor as federation layer for distributed resources
- XRootD as data access protocol

Repositories

- Main repo: https://github.com/comp-dev-cms-ita
- Dask:
  - Extension: https://github.com/comp-dev-cms-ita/dask-labextension
  - Jobqueue: https://github.com/comp-dev-cms-ita/dask-remote-jobqueue
- Analysis example: https://github.com/comp-dev-cms-ita/analysis-examples

AuthN/Z: a “token native” system

We thought the system to be **token based**

- In other words: **from IAM @CMS → JupyterHUB/HTCondor/DASK reverse proxy**

Although a bit ahead of time, this comes with a significant benefit when we need to automatically define what resources can access who in a dynamic/modern fashion

So far in HTCondor we are using the “good old” *mapfile* to match user IAM id with condor user. This is in evolution though, **possibly toward capability based access** (in any case, **no changes would be needed to the infrastructure**)

```
apiVersion: v1
data:
  condormapfile: |
    SCITOKENS  https:\/\/dodas-iam.cloud.cnaf.infn.it\/,1e7074e5-96fe-43e8-881d-4d572c128931 dciangot
    SCITOKENS  https:\/\/dodas-iam.cloud.cnaf.infn.it\/,d0203717-30ad-407e-ab82-a48b93baed57 vpadulan
    SCITOKENS  https:\/\/cms-auth.web.cern.ch\/,51e0c369-345a-46b1-812f-61a4269cda8f  etejedor
    SCITOKENS  https:\/\/cms-auth.web.cern.ch\/,0c765e2d-993e-4ed2-abf8-9c54dcc34546  ttedesch
    SCITOKENS  https:\/\/cms-auth.web.cern.ch\/,4619b517-39d1-4b76-87df-69cbb15a0dee  mtracoll
    SCITOKENS  https:\/\/cms-auth.web.cern.ch\/,78f275d5-bb1a-4b2d-9956-f82316a8482e  spiga
    SCITOKENS  https:\/\/cms-auth.web.cern.ch\/,c4b22a94-6dda-4312-b77f-726813e963ae  dciangot
    PASSWORD (*.) condor
    GSI (.*) anonymous
```

Recap

So far we **aim to reduce the amount of "in house" solutions** and we focus on tuning and scaling the described system.

In summary, we came up with a testbed setup to provide a playground for the design of a future analysis infrastructure

- **Leveraging state-of-the-art software tool sets**
  - User accesses through JupyterHub
  - User works within JupyterLab
- **Develop locally than scale out** and make use of ***already-available/spare*** **resources**
  - Access to resources using Dask
    - Dask Scheduler to manage tasks
  - XRootD to manage data

Already challenging enough, but we indeed think that the value lives in doing **benchmark of all of this with real use cases.**

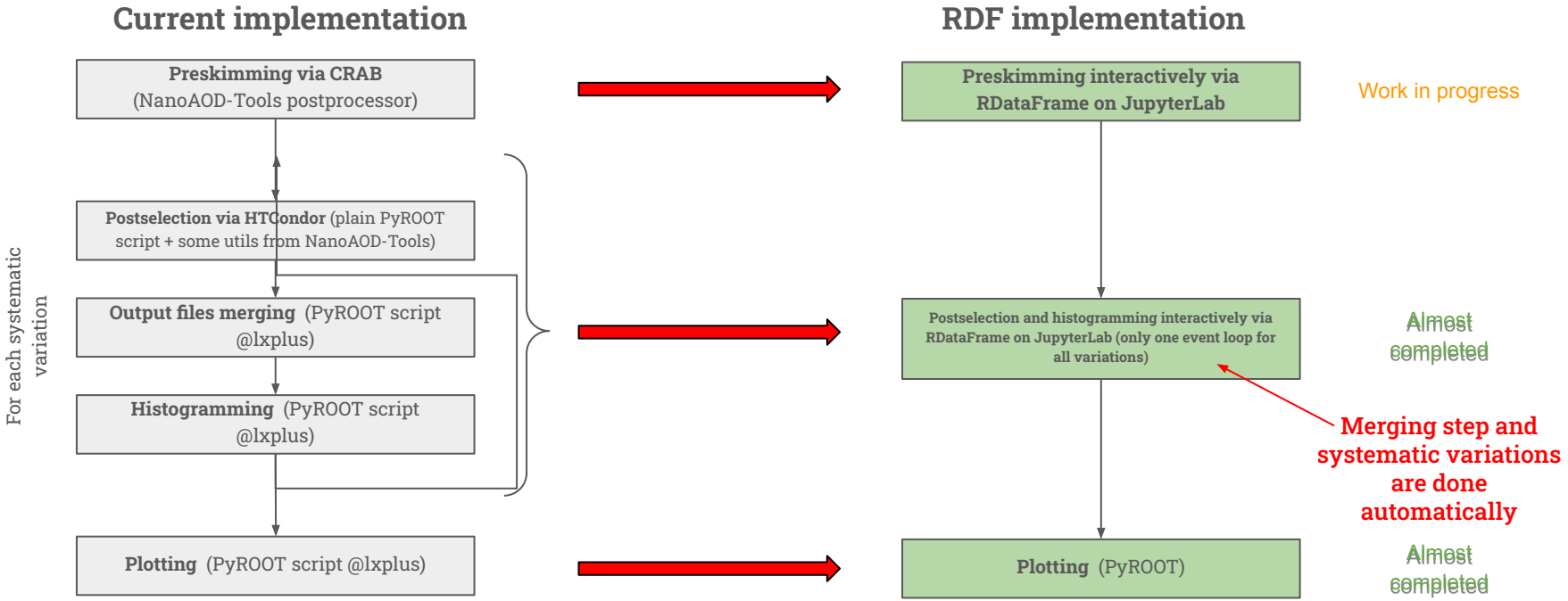
Analysis porting

Currently, porting **VBS SSWW with a lepton and an hadronic tau in final state to RDF:**

- Data to be processed: **ca. 6TB (Data + MC ReReco samples for 2017 and 2018)**
- Once the whole porting completed: **validation** and **benchmarking.**

A special thanks go to RDF developers

Analysis porting



Recap of main features

- The language of choice is **Python**
  - while still need to be interfaced with C++ for complex and heavy duties
- Heavy focus on NanoAODs
  - In theory not limited to, though
- Support for **distributing processing seamlessly** (no code changes needed) on:
  - **multiple threads**
  - **multiple cores** on distributed systems
- Under the hood
  - **The framework will take the user requests and split the payloads** in the most efficient way
  - It will **re-collect back** all the processed pieces **automatically**