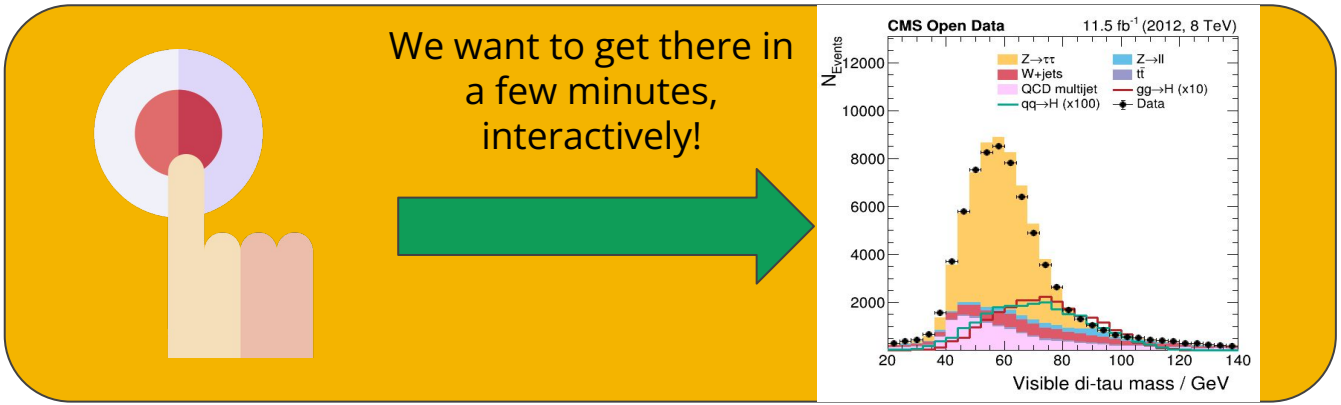
Interactive distributed computing for HEP on multi-managed cluster resources

Vincenzo Eduardo Padulano, Stefan Wunsch, Rene Caspart, Max Fischer, Manuel Giffels

HEP analysis in analysis facilities

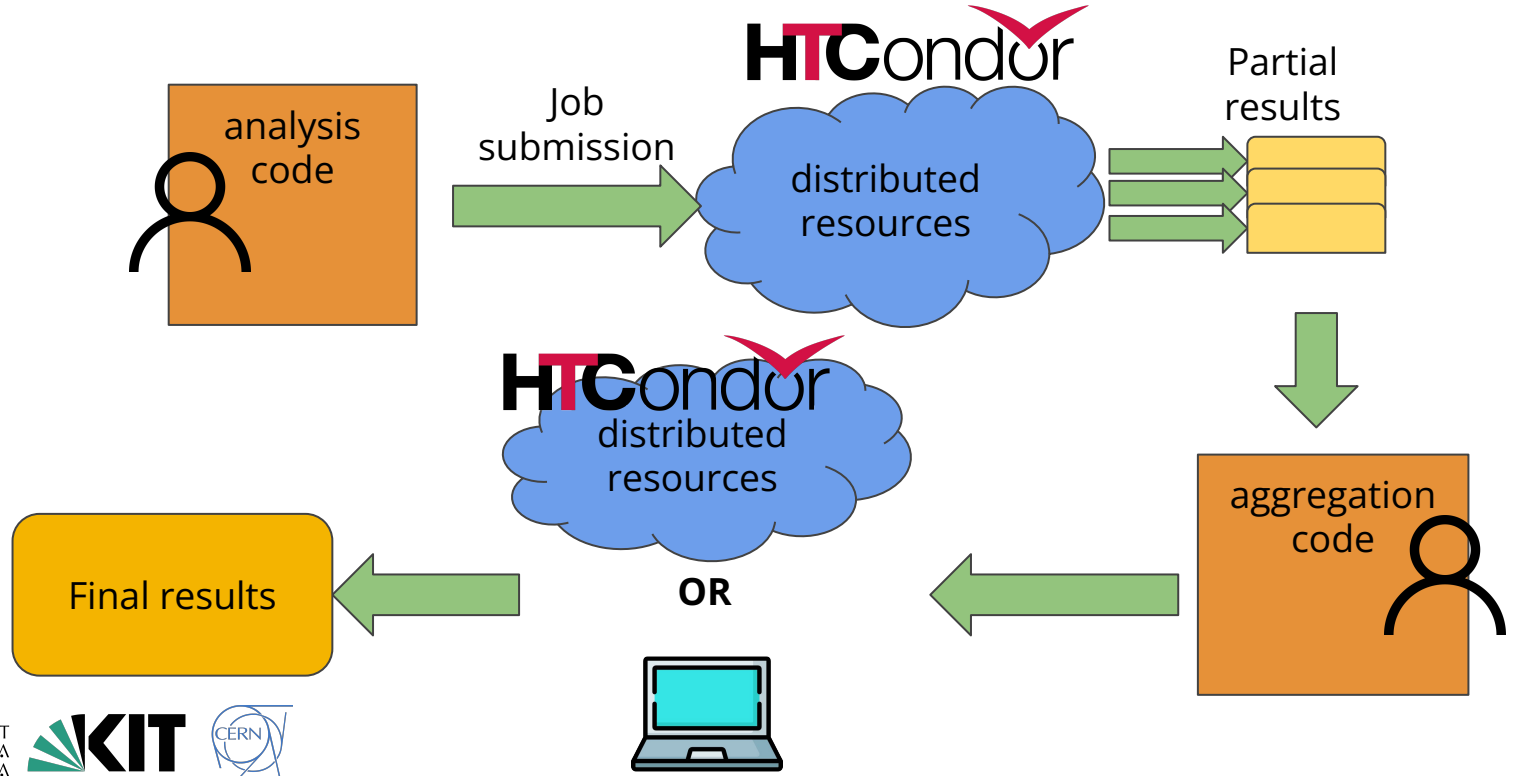
Analysis facility: Local computing cluster at a **university**, **T3** center in the **WLCG**:

- **Limited budget**, must make use of all resources at all times
- Cluster resources managed through **batch systems** and job queues
- **High** computing power **needs**: run analysis **whenever** needed, high variability in runtime (from minutes to **hours**)



Batch computing in HEP

Traditional HEP distributed computing: **high-latency** job queues and **multiple steps** of user interaction



Distributed computing with ROOT

Target

- Final physics analysis steps
  - Making final data filtering, **histograms**, …
  - Investigate physics data, **interactively**
  - Repeating the same analysis while **tweaking** the parameters

Objective

- Scale out resources with **minimal latency**
  - Enable **interactive large scale** data analysis
  - **Automatic** result **aggregation** directly in the application
  - Minimal turnaround cycle
    → Maximal efficiency for the analysts

Why is this not trivial?

- Most common resource and application scheduler in HEP computing ecosystem
- Users submit their jobs in long queues, preventing interactiveness
- Targets efficient job scheduling maximizing the resource utilization
  → **Cost efficient**

- Targets **interactive** workload
- Small task granularity (~ 1MB) allows to **scale out** smaller workloads with **low latency**
- Bad resource utilization if no user performs analysis, e.g. due to daytime working hours
  → **Expensive**

Can we still have the best of both worlds?

Idea

1. Run **HTCondor** as the resource **manager** on the **cluster**
2. Get from HTCondor resources quickly if a **user** submits a Spark **application**
3. **Register** these **resources** on the Spark backend side (**YARN**)
4. Share the newly allocated resources of the Spark cluster between **multiple users** to minimize further latency
5. **Give resources back** to HTCondor if the utilization of the Spark cluster drops again

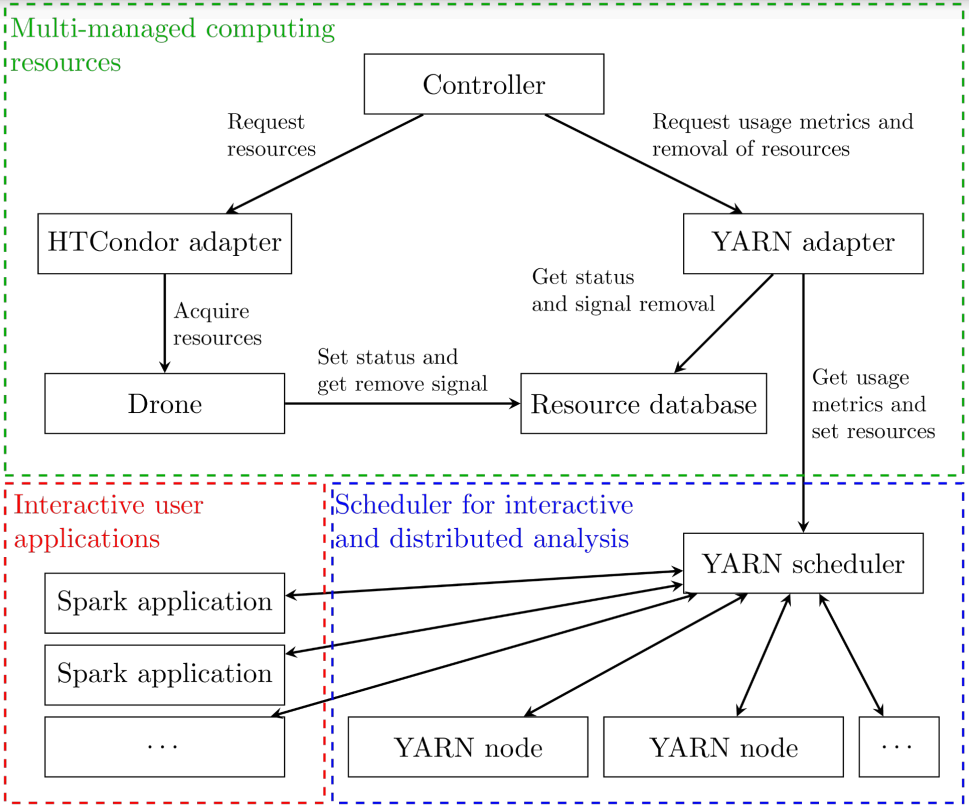
This mechanism needs a coordination layer!

COBaID/TARDIS

- A dynamic resource integration system, developed at KIT
- In our use case:
  - It is a **service** running alongside of HTCondor and Spark/YARN
  - It is able to **monitor** and **control** utilization and allocation of physical resources

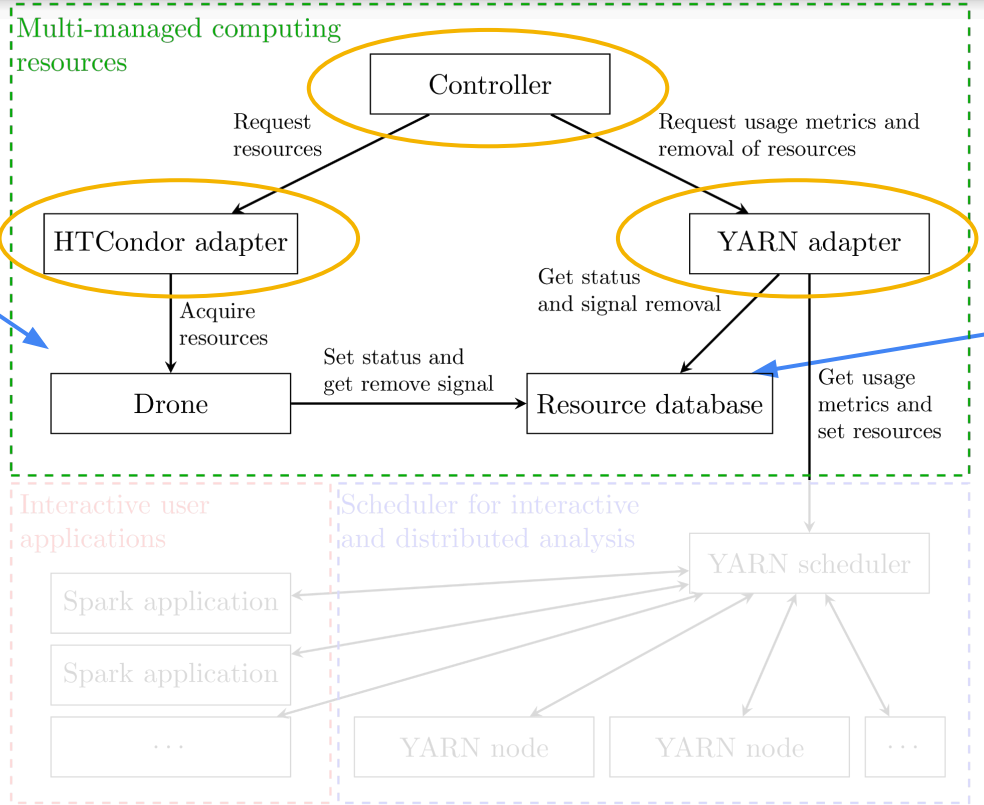
- Our first (basic) approach at coordinating the cluster resources:
  - **Start good things**
    If **user** applications are running, **allocate** resources to Spark
  - **Stop bad things**
    Remove idle resources from Spark and **give** them back to HTCondor

Design overview



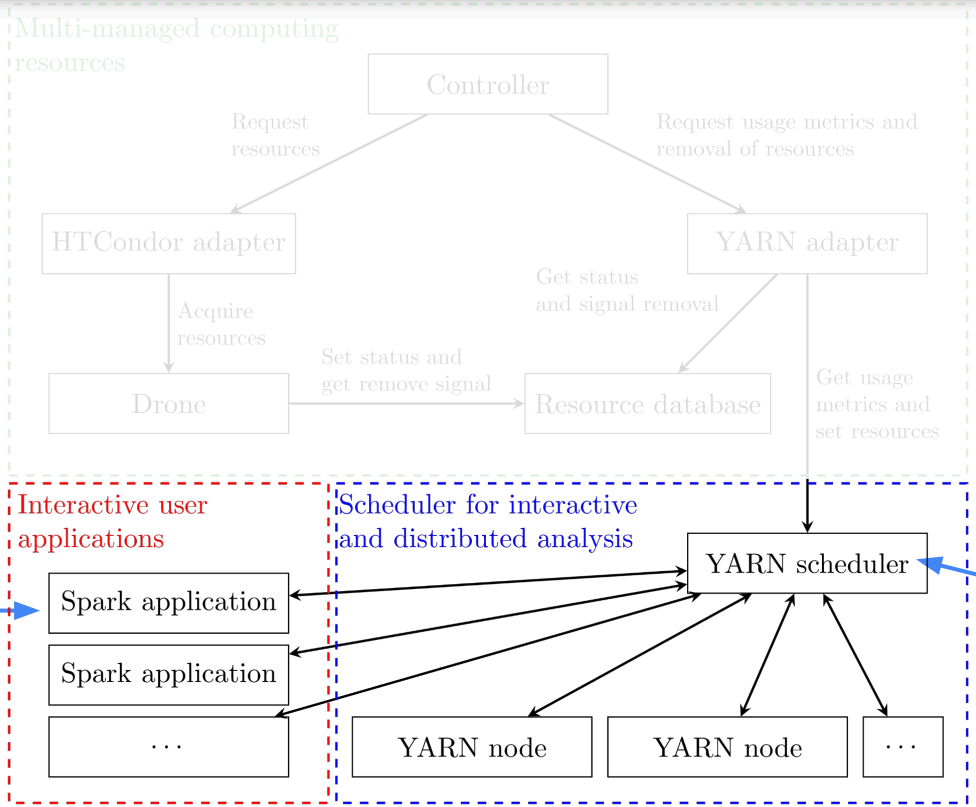
Design overview

Resource allocation with HTCondor



Sync resource status between components

Design overview

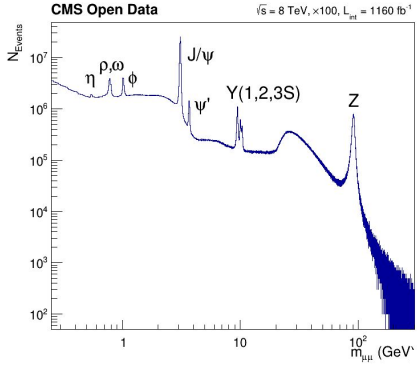


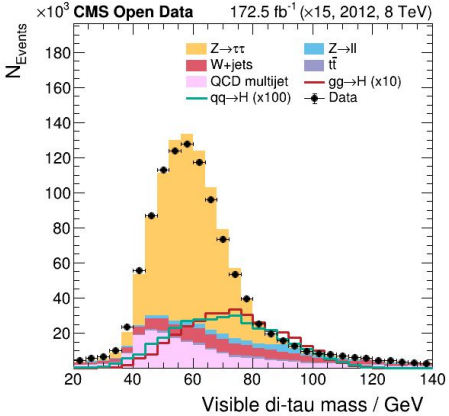
Users interface only with Spark

YARN controls only its resource quota.

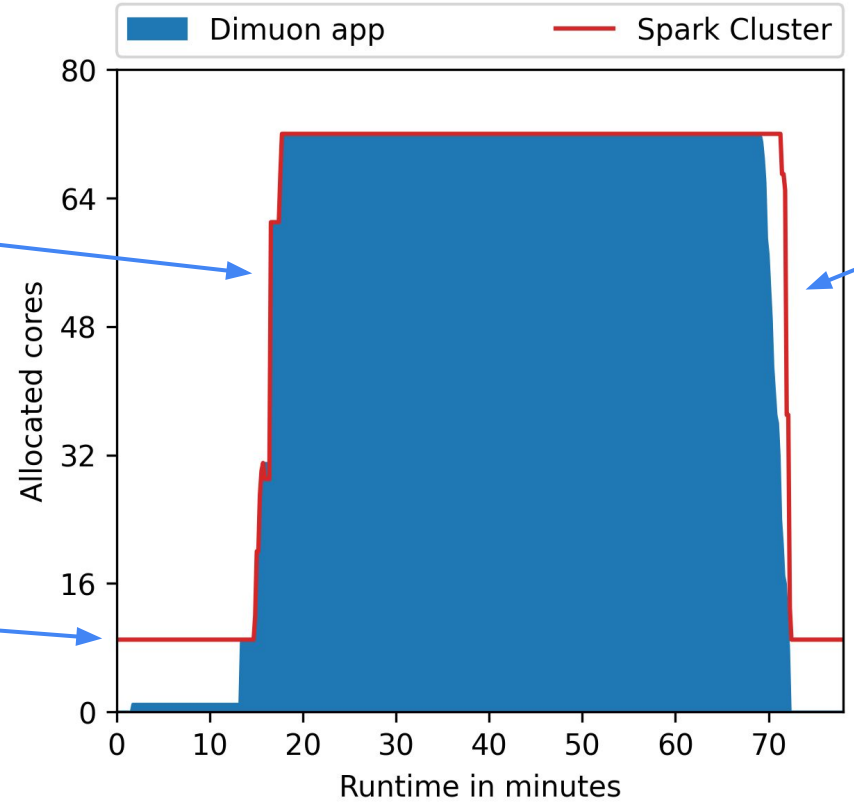
Open Data Analyses

- Open source and open data freely accessible via the CERN Open Data portal
  - Open benchmarks are crucial to compare software solutions
- Dimuon spectrum
  - 2.1 TB (x1000 of the original size)
  - 1000 ZLIB compressed files
  - All data is read in the analysis
- HiggsTauTau analysis
  - Total size 68 GB
  - Input: 9 NanoAOD-like datasets
  - 70% of the data is deserialized





Single Dimuon app



Resources from HTCondor at app request

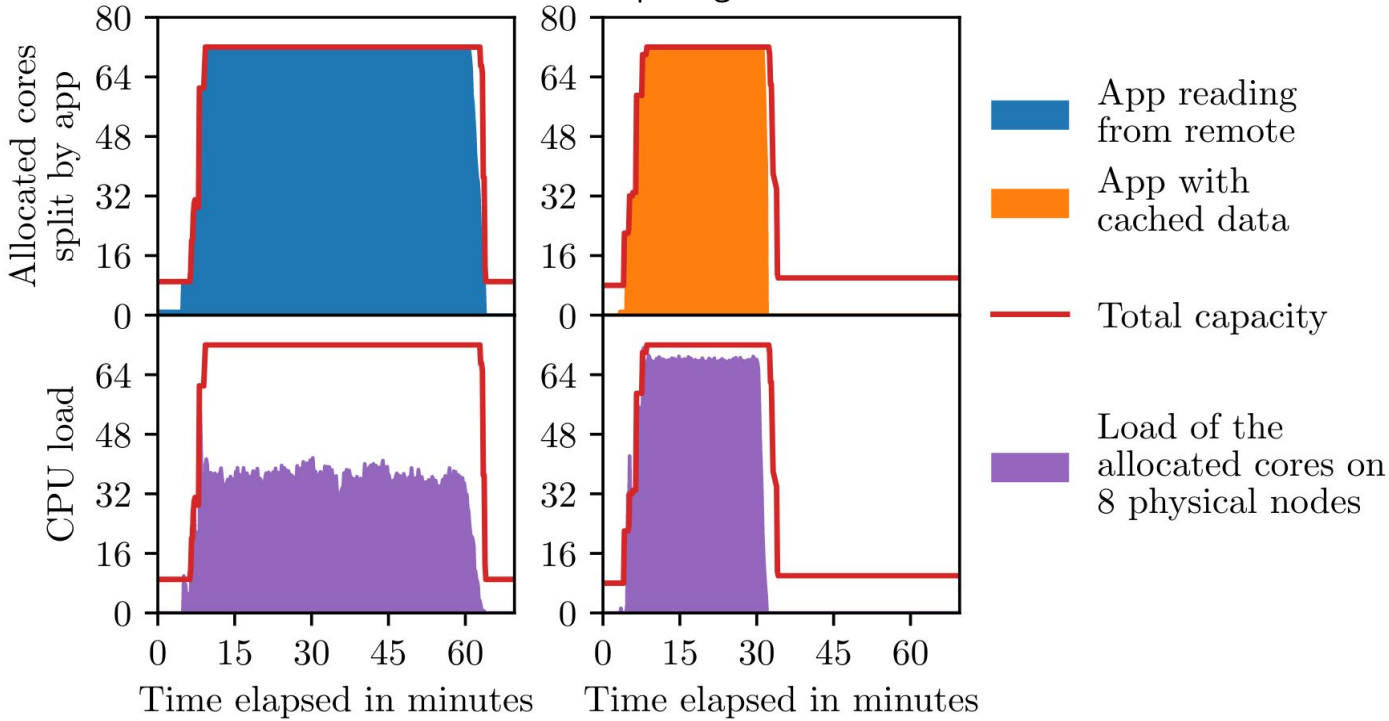
Final deallocation, back to HTCondor

Initial Spark cluster capacity. No impact on resource usage

Single Dimuon app: caching

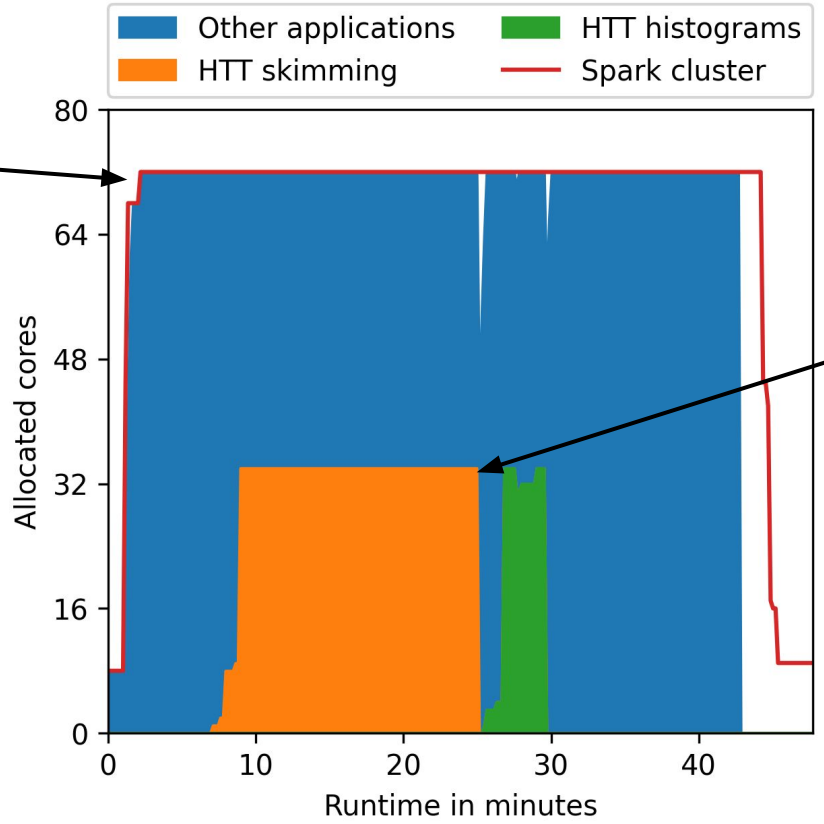
Left side: 1000 files read from EOS

Right side: 1000 files read from the computing nodes local SSD



Multiple apps example

YARN manages its resource quota



FAIR sharing of resources

Summary

1. Traditional distributed computing in HEP suffers from **high latency** and asks the user to take care of **multiple steps** of the process

2. **Interactive** distributed analysis is **possible** with ROOT + **Spark** (and potentially other systems)

3. Analysis facilities run on limited budget, it is **impossible** to have resources always dedicated to Spark since that would mean a lot of **idling time**

4. Let's create a system that allows for interactive applications **without wasting** precious resources

5. This design implements a **resource coordination** layer on top of a **multi-managed** cluster with HTCondor and YARN

Future steps

1. Further **testing**: more users, more cores, more analyses
2. Try different **policies** for the **coordination** layer
3. Include **authentication** for data with access restrictions
4. Deployment on other facilities (soon tests at **TOPAS** T3 at KIT)

Follow the developments on our [github repo]