A Prototype U.S. CMS analysis facility

Mat Adamec, Ken Bloom, **Oksana Shadura,**
University of Nebraska, Lincoln

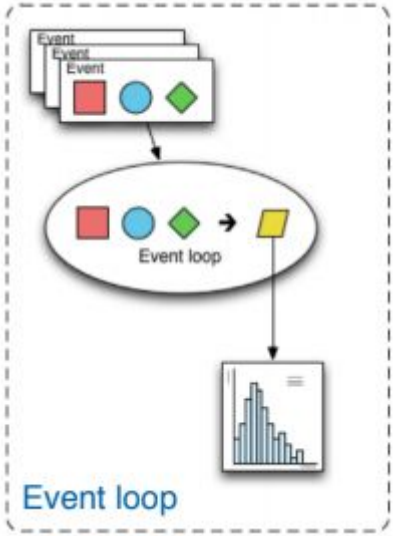
Garhan Attebury, Carl Lundstedt, Derek Wietzel
University of Nebraska Holland Computing Center

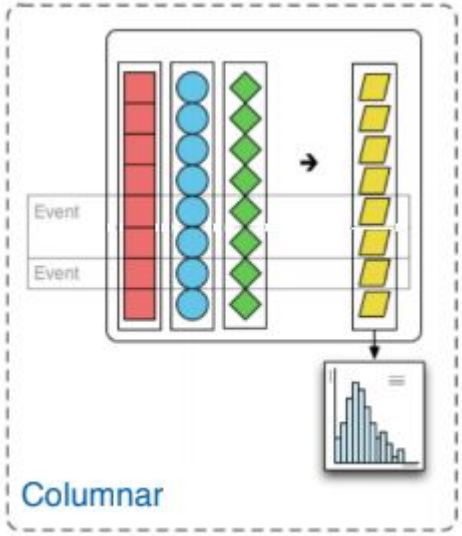
Mátyás Selmeci
University of Wisconsin, Madison

Brian Bockelman
Morgridge Institute

coffea - Columnar Object Framework For Effective Analysis

- Leveraging large data and data analysis tools from Python to provide an array-based syntax for manipulating HEP event data
- Stark contrast to well established event loop techniques
- **Tremendous potential to fundamentally change the time-to-science in HEP**
- **Scales well horizontally**
- Cannot easily utilize current analysis facilities (T2s) as the analysis is not grid friendly, it's meant to be quasi-interactive

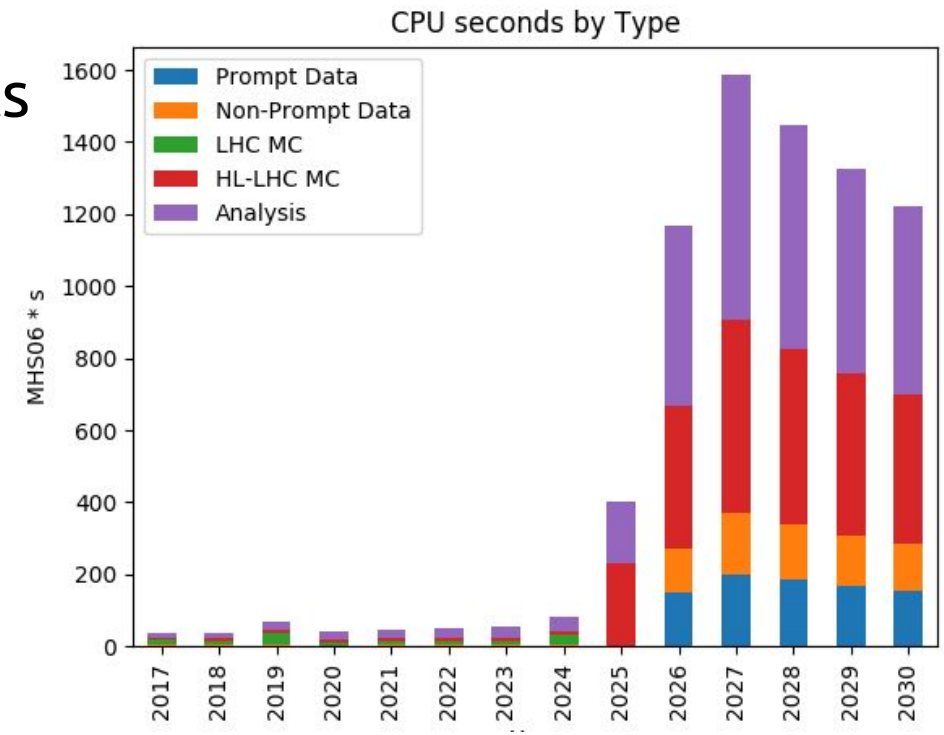




coffea: performance challenges

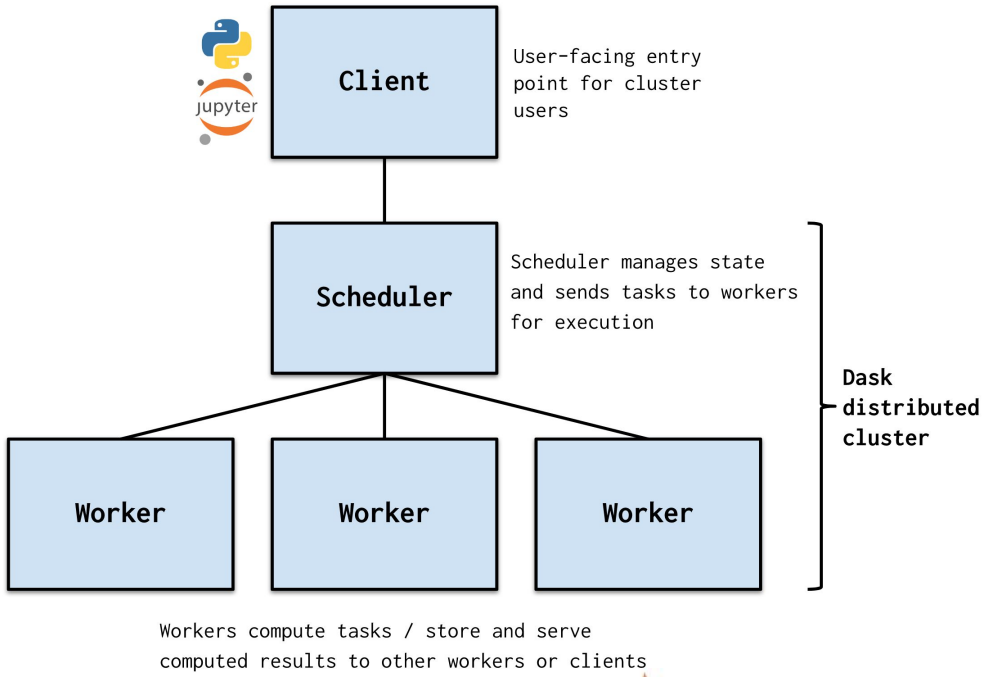
- The Present Challenge
  - Analyze all LHC Run 2 data: O(10 billion events)
  - Investigate data quality issues with fast time-to-insight
  - Optimize complex algorithms (e.g. deep learning algorithms)
- Multiply by O(1000) data analysts
- These challenges magnified 20x in HL-LHC

"coffea - Columnar Object Framework For Effective Analysis", Nick Smith, CHEP 2019



Dask: scalable analytics in Python

- Dask provides flexible library for parallel computing in Python
- *Think of Dask as run-time parallel + cluster plugin for Python*
- Easily installed via Conda as the module “distributed”
- NOT really designed with multi-user environments in mind out-of-the-box
- Integrates with HPC clusters running a variety of schedulers including SLURM & HTCondor via “dask-jobqueue”



Requirements for Analysis Facility @ T2

- Easy to use for users
- Scalable (dynamically/automatically)
- Responsive/Interactive
- Utilize currently deployed hardware/middleware
- Minimally intrusive for site administrators
- Get work ('effort' & CPU) accounted for by CMS

Usage Patterns Are Changing
Resources Should Change, Too

Proposed Analysis Facility @ T2 Nebraska

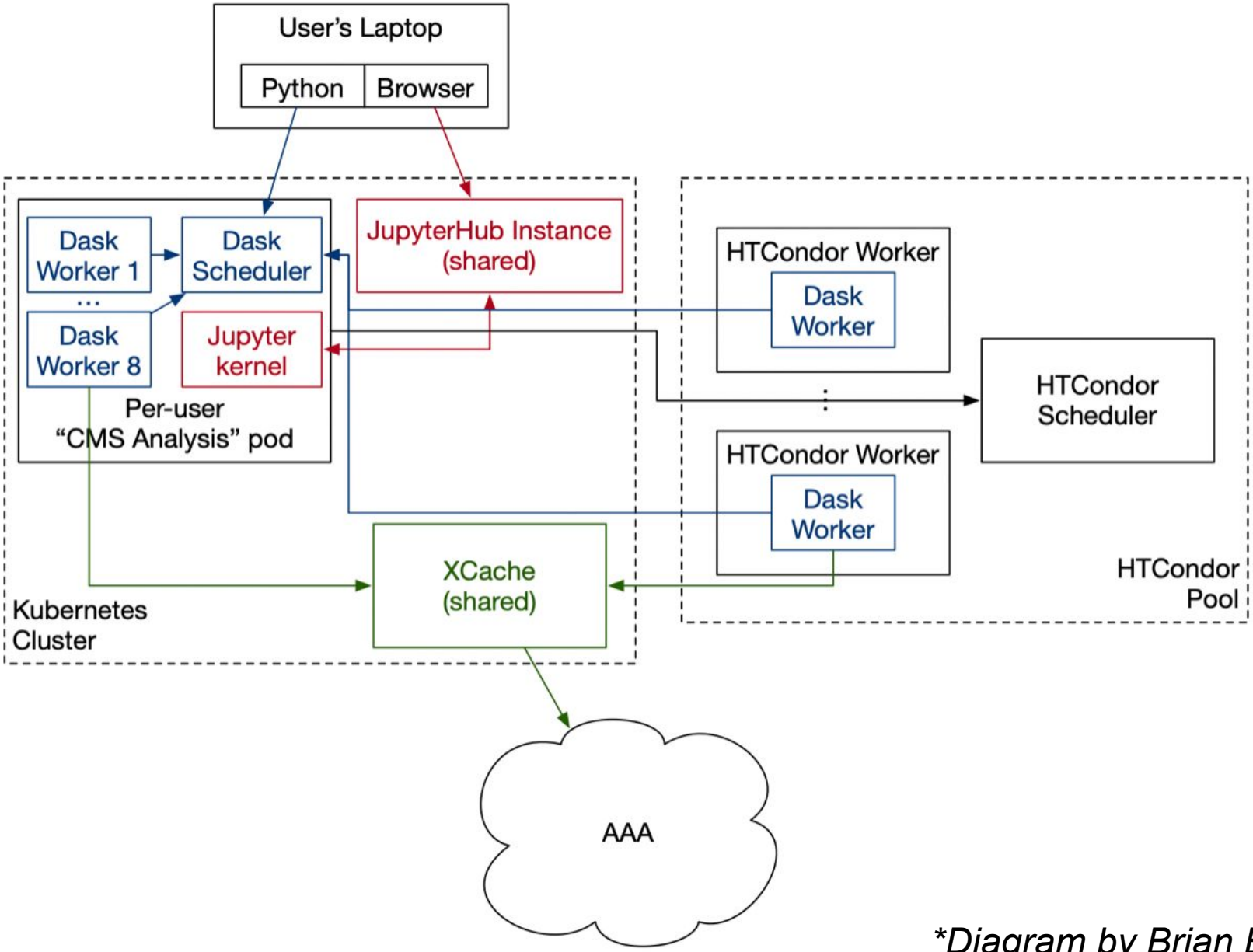
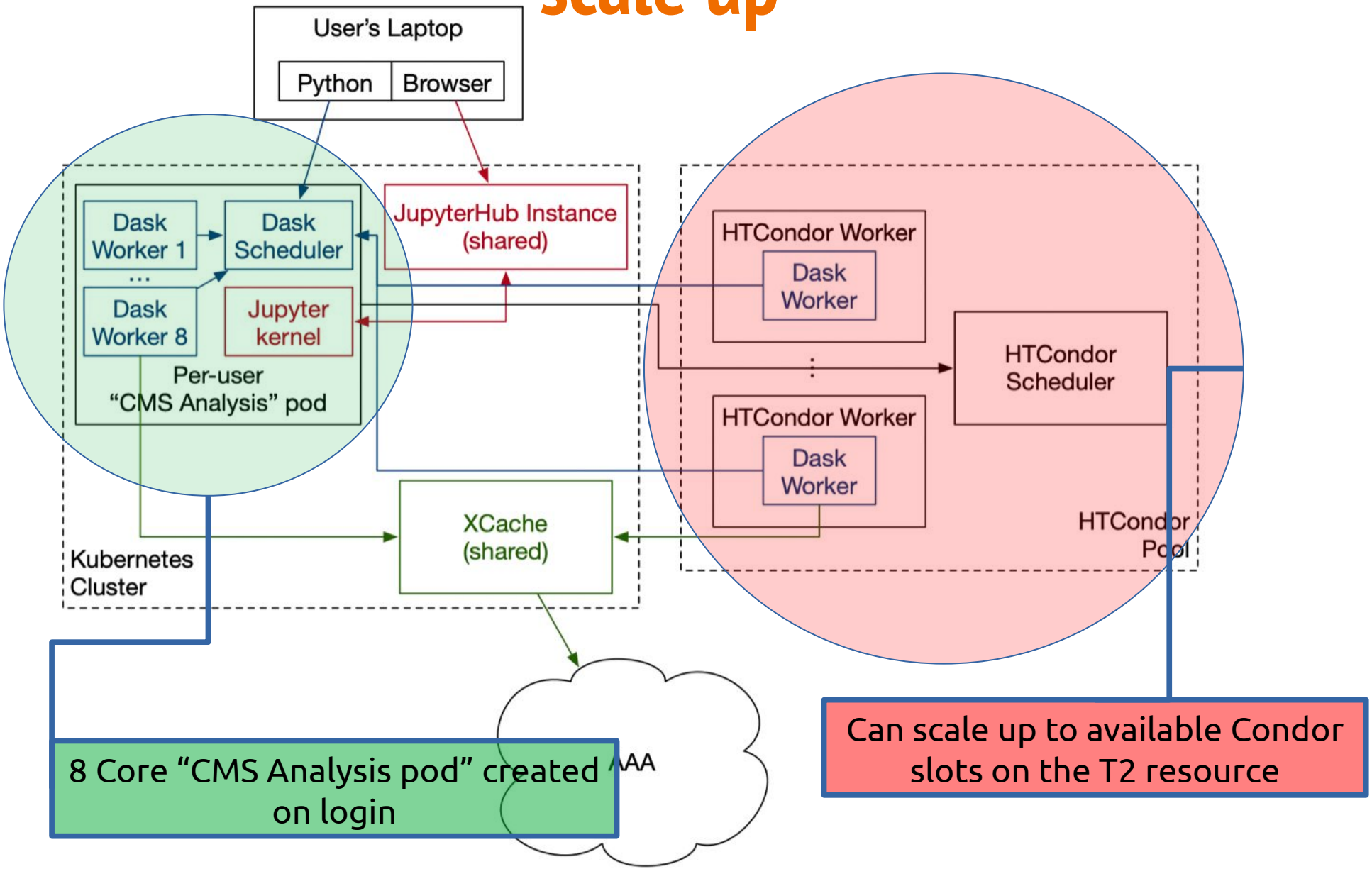
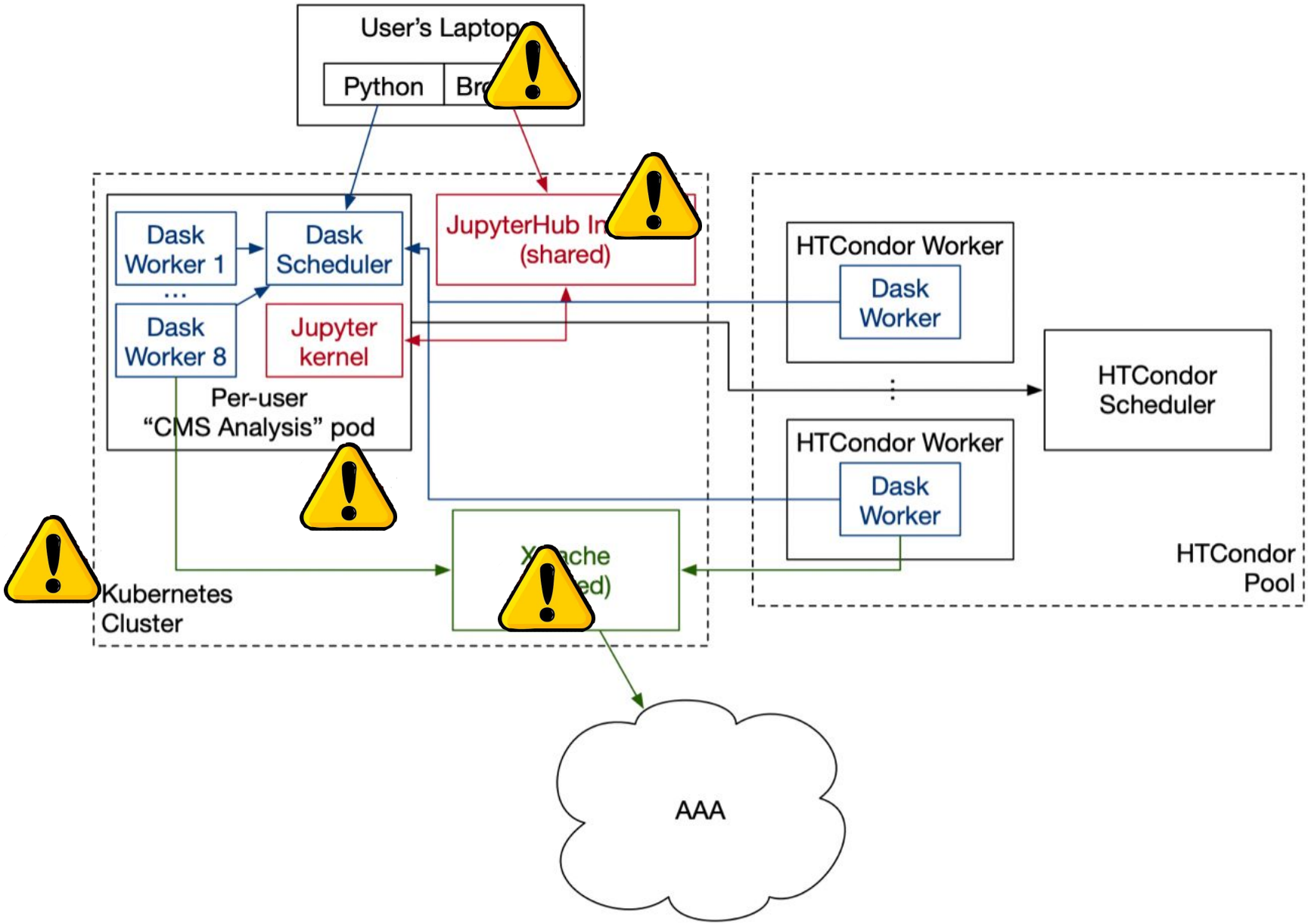


*Diagram by Brian Bockelman

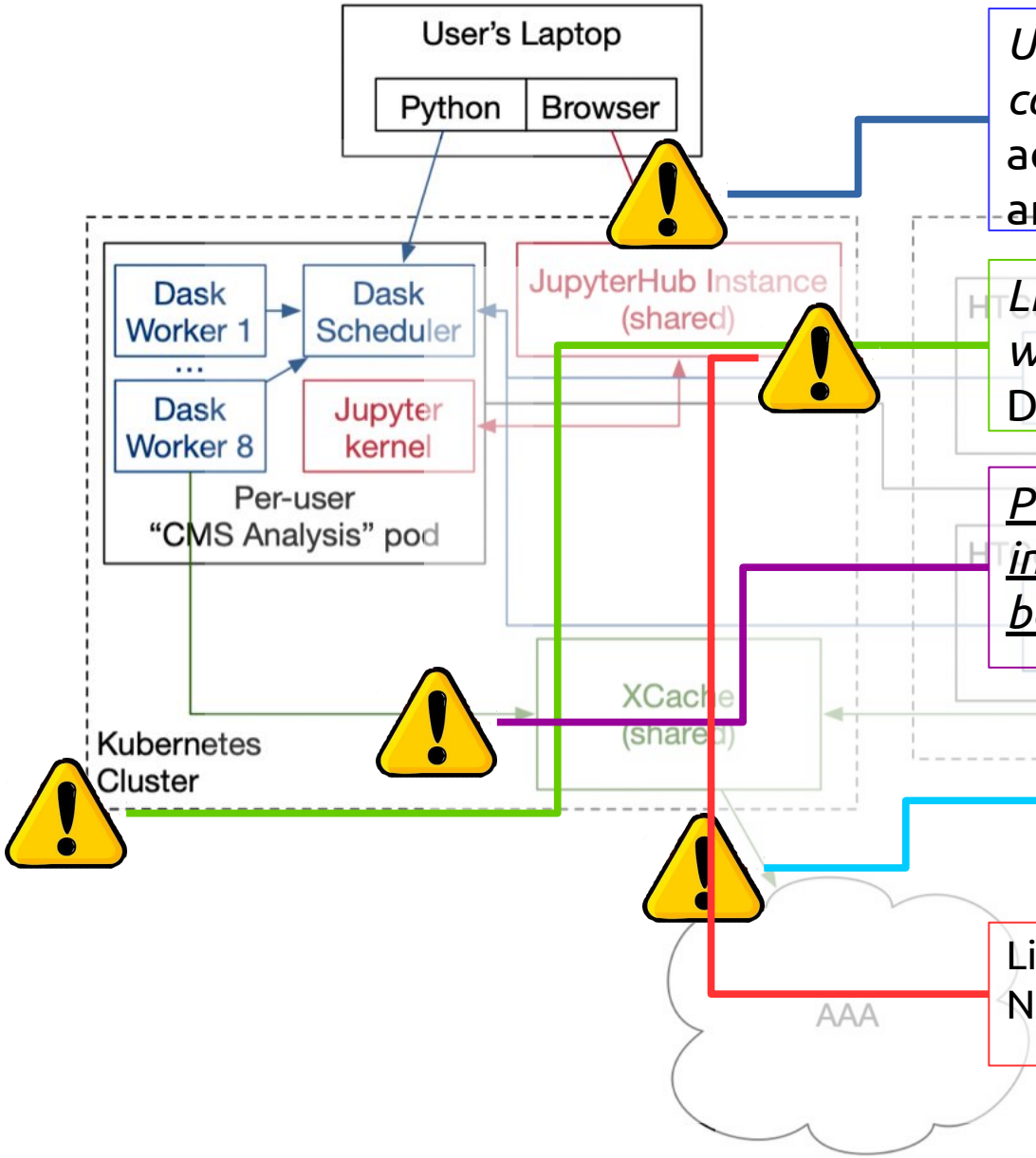
Proposed Analysis Facility @ T2 Nebraska: scale up



Proposed Analysis Facility @ T2 Nebraska: challenges



Proposed Analysis Facility @ T2 Nebraska: challenges



User Management at the T2s is (and will continue to be) a non-starter. User access management will have to be done at another administrative level (SSO)

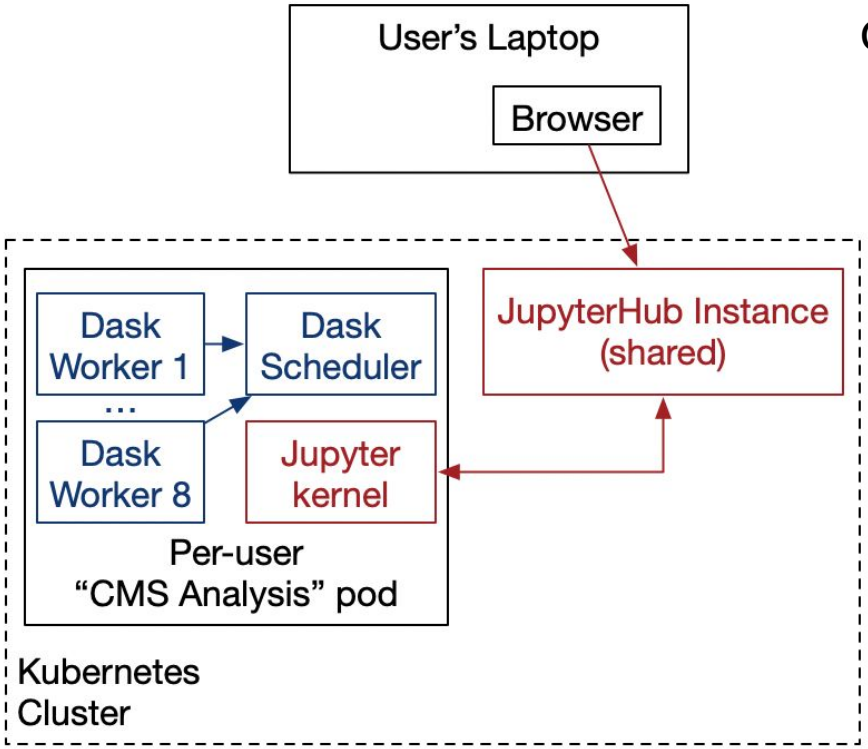
Limited experience with Kubernetes (solid experience with Docker.)

Performance can be sensitive to caching, integration and performance studies need to be done long term.

Not clear how well things will perform using AAA + batch.

Little to no operational experience at Nebraska T2 for JupyterHub management.

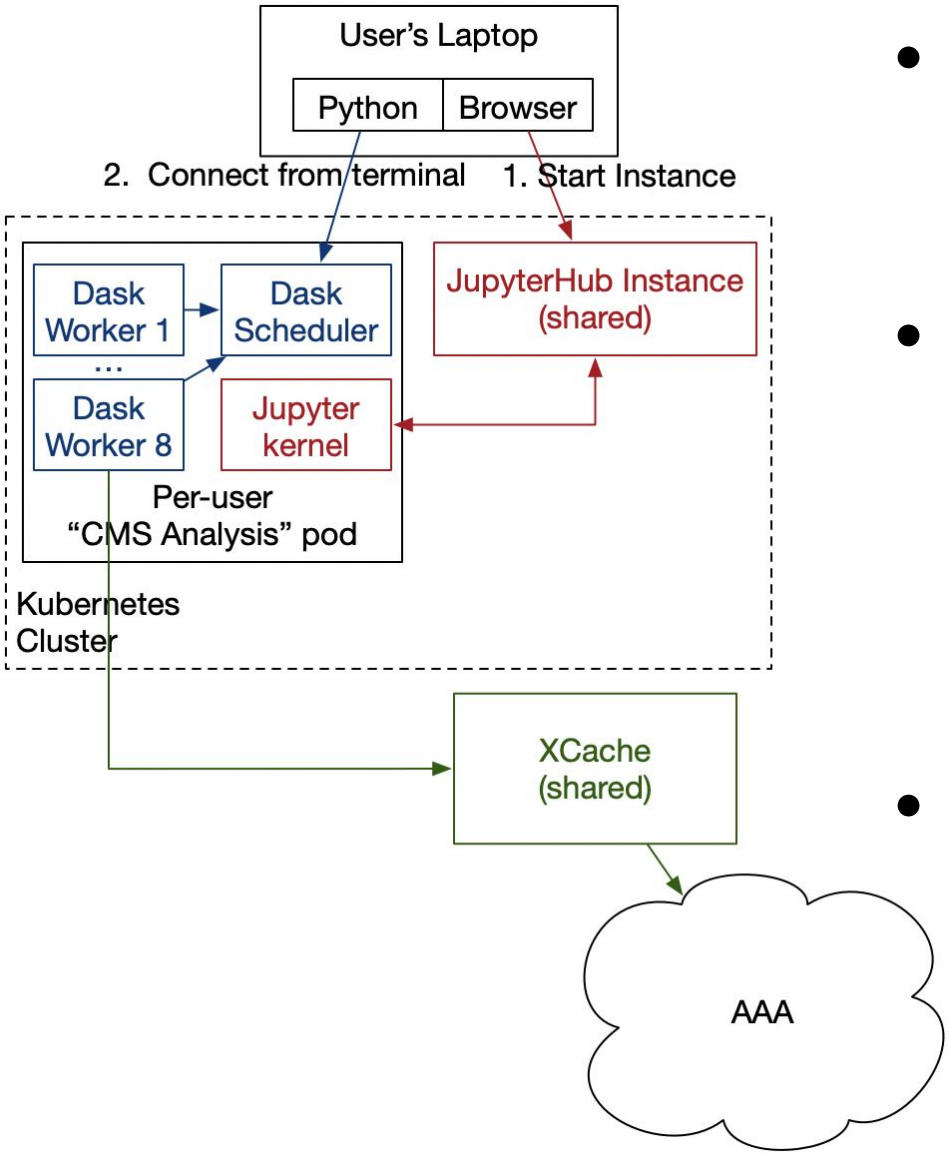
Analysis Facility @ T2 Nebraska



Configuration:

- *Enabled token authentication* in Condor infrastructure
- *Setuped a Kubernetes (k8s) cluster* and use the "*Zero 2 JupyterHub" (z2jh)* project to put together a basic JupyterHub instance
- Developed a *highly customized "CMS Analysis" container* with all the necessary dependencies
- JupyterHub uses the KubeSpawner to create new pods. We utilize a *pod customization hook to create secrets and services*:
  - **Pod can expose the Dask scheduler to the outside world**
  - **Pod can authenticate with services like HTCondor and XRootD**

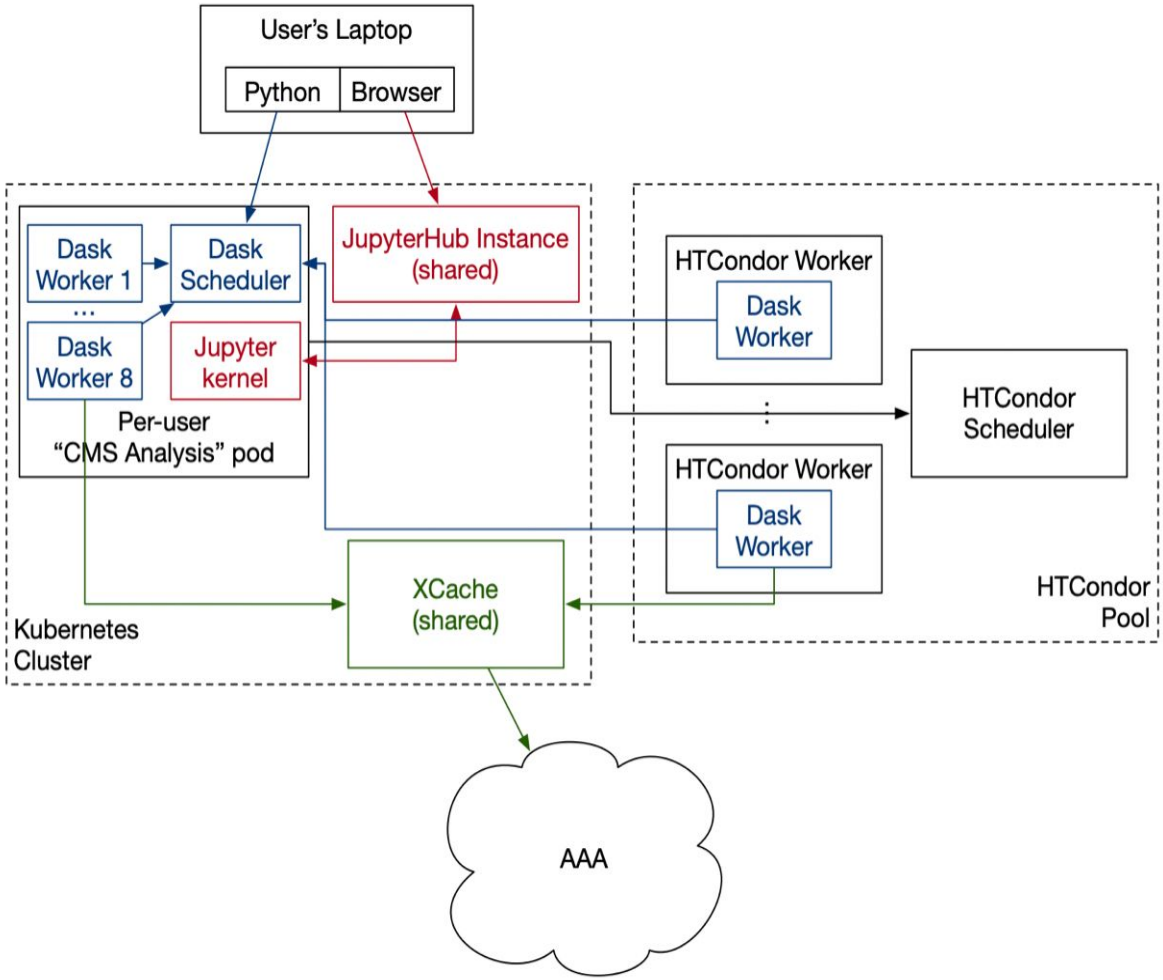
Analysis Facility @ T2 Nebraska



- Integration of XRootD
  - Each pod's unique secret includes an **auto-generated macaroon** *authorizing the pod to access files at the site XCache server*
- Developed a custom XRootD client plugin so whenever the prefix root://xcache/ is used, then:
  - The hostname is replaced with the correct one for the local site (using environment variables)
  - Token authorization is automatically used & embedded in the URL.
- A custom XCache container was made to make GSI auth optional and allow token auth after an anonymous login

  See the plugin code: https://github.com/bbockelm/xrdcl-authz-plugin

Analysis Facility @ T2 Nebraska



- Finally, we use a minimally-patched version of the *HTCondorCluster integration from Dask* to allow auto-scaling out to the local HTCondor pool
  - ***Security:*** *TLS enabled communication between workers and scheduler*
- Jobs run in the container on the HTCondor worker node; HTCondor exposes an incoming port to provide the necessary connectivity.
- All of this is being incorporated into a *Helm chart* -- many rough edges, but **it can eventually be portable to other sites**

CMSAF @ UNL Setup

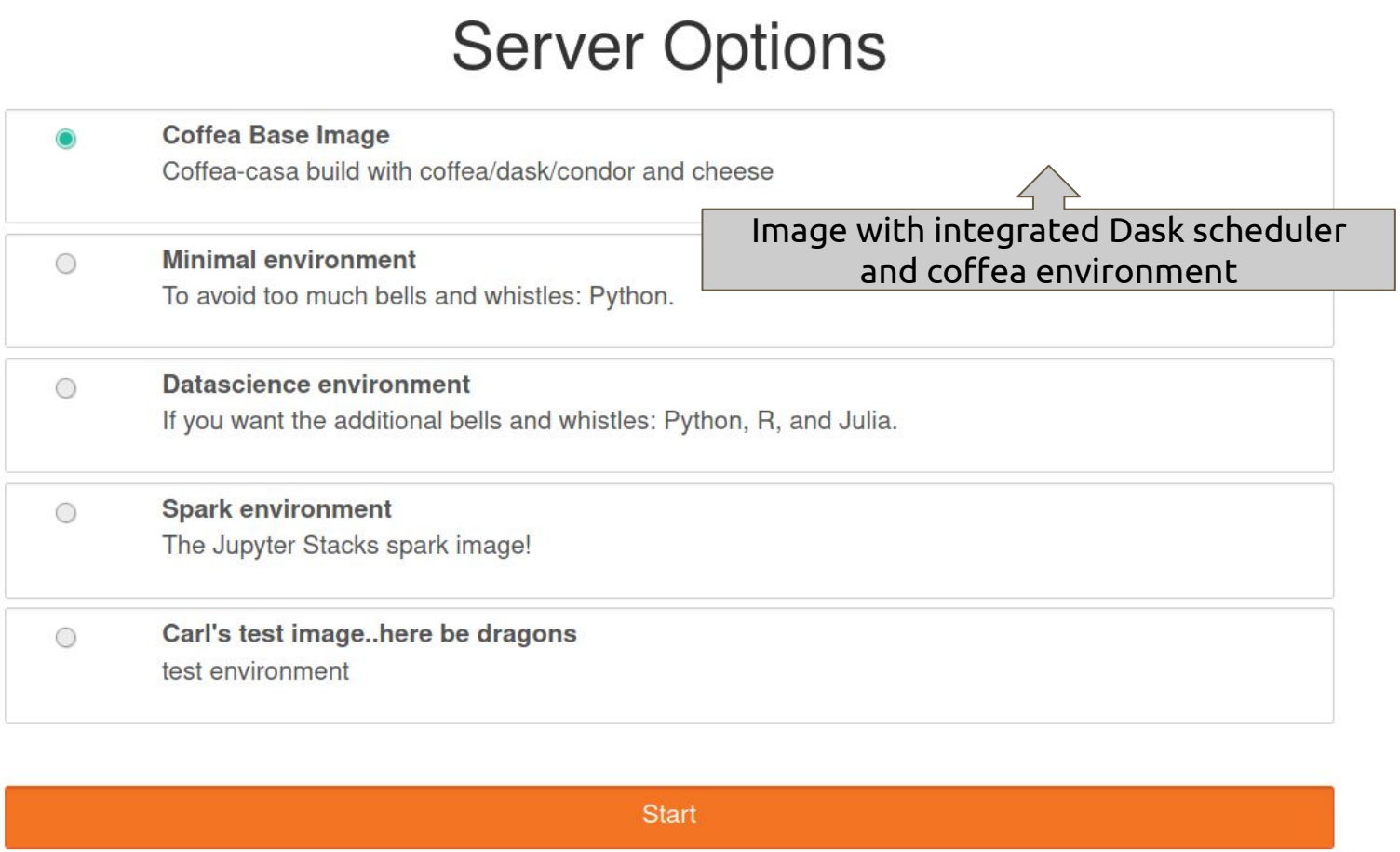
- JH setup: https://github.com/CoffeaTeam/jhub (except specific secrets)
- Docker images for Dask Scheduler and Worker: https://github.com/CoffeaTeam/coffea-casa
  - https://hub.docker.com/r/coffeateam/coffea-casa
  - https://hub.docker.com/r/coffeateam/coffea-casa-analysis
- Docker image for JupyterHub (to get macaroons in the launch env) https://github.com/clundst/jhubDocker

JupyterHub + JupyterLab + Dask setup @ UNL

- JH is launched using Helm charts (together with users secrets)

Login

- Docker image starting JupyterLab is integrated with HTCondor Dask Scheduler communicating with T3
  - Powered by Dask Labextention, which is integrated in the Docker image



CMSAF @ UNL Analysis

```
[1]: import os
     import dask
```

```
[2]: from distributed.security import Security
     from coffea import hist
     from coffea.analysis_objects import JaggedCandidateArray
     import coffea.processor as processor
```

```
[3]: from dask.distributed import Client, LocalCluster
     from dask_jobqueue import HTCondorCluster
     from dask_jobqueue.htcondor import HTCondorJob
```

```
[4]: from coffea_casa.coffea_casa import CoffeaCasaCluster
```

```
[5]: fileset = {
         'Jets': { 'files': ['root://eospublic.cern.ch//eos/root-eos/benchmark/Run2012B_SingleMu.root'],
                   'treename': 'Events'
                 }
     }
```

```
[6]: # This program plots an event-level variable (in this case, MET, but switching it is as easy as a dict-key change). It also demonstrates an easy use of the book-keeping cutflow tool, to keep track of the number of events processed.
     # The processor class bundles our data analysis together while giving us some helpful tools.  It also leaves looping and chunks to the framework instead of us.
     class METProcessor(processor.ProcessorABC):
         def __init__(self):
             # Bins and categories for the histogram are defined here. For format, see https://coffeateam.github.io/coffea/stubs/coffea.hist.hist_tools.Hist.html && https://coffeateam.github.io/coffea/stubs/coffea.hist.hist_tools.Bin.html
             self._columns = ['MET_pt']
             dataset_axis = hist.Cat("dataset", "")
             MET_axis = hist.Bin("MET", "MET [GeV]", 50, 0, 100)
             # The accumulator keeps our data chunks together for histogramming. It also gives us cutflow, which can be used to keep track of data.
             self._accumulator = processor.dict_accumulator({
                 'MET': hist.Hist("Counts", dataset_axis, MET_axis),
                 'cutflow': processor.defaultdict_accumulator(int)
             })

         @property
         def accumulator(self):
             return self._accumulator

         @property
         def columns(self):
             return self._columns

         def process(self, df):
             output = self.accumulator.identity()
             # This is where we do our actual analysis. The df has dict keys equivalent to the TTree's
             dataset = df['dataset']
             MET = df['MET_pt']
             # We can define a new key for cutflow (in this case 'all events'). The
             output['cutflow']['all events'] += MET.size
             output['cutflow']['number of chunks'] += 1
             # This fills our histogram once our data is collected. Always use .fla                                                                e bin.
             output['MET'].fill(dataset=dataset, MET=MET.flatten())
             return output

         def postprocess(self, accumulator):
             return accumulator
```

Custom method *CoffeaCasaCluster* (on top of dask_jobqueue.HTCondorCluster) to deploy Dask on common HTCondor job queue with possibility to specify Dask worker image, TLS and enabled auto scaling (it is reusable on the any other facility)

```
[7]: HTCondorJob.submit_command = "condor_submit -spool"
```

```
[8]: host_ip = os.getenv("HOST_IP")
```

```
[9]: client = CoffeaCasaCluster(worker_image="coffeateam/coffea-casa-analysis:0.1.37", external_ip=host_ip, min_scale=5, max_scale=6, tls=True)
```

```
distributed.scheduler - INFO - Clear task state
distributed.scheduler - INFO -   Scheduler at: tls://192.168.14.175:8787
distributed.scheduler - INFO -   dashboard at:                    :8786
distributed.scheduler - INFO - Receive client connection: Client-d7f7b500-beca-11ea-86be-7a45415d17d5
distributed.core - INFO - Starting established connection
```

Starting scheduler...

Starting Dask workers over HTCondor queue with enabled autoscaling

[10]:
```
exe_args = {
        'client': client,
    }
```

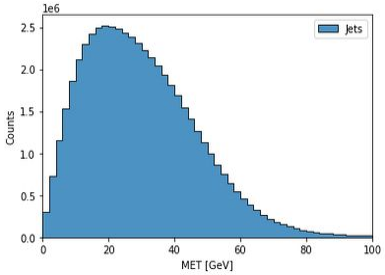
[11]:
```
output = processor.run_uproot_job(fileset,
                                  treename = 'Events',
                                  processor_instance = METProcessor(),
                                  executor = processor.dask_executor,
                                  executor_args = exe_args
                                  )
```

```
[                                        ] | 0% Completed |  0.9s
distributed.core - INFO - Event loop was unresponsive in Scheduler for 5.96s.  This is often caused by long-running GIL-holding functions or moving large chunks of data. This can cause timeouts and instability.
[                                        ] | 0% Completed | 44.5s
distributed.scheduler - INFO - Register worker <Worker 'tls://129.93.182.135:32814', name: htcondor--3436325.0--, memory: 0, processing: 1>
distributed.scheduler - INFO - Starting worker compute stream, tls://129.93.182.135:32814
distributed.core - INFO - Starting established connection
[                                        ] | 0% Completed | 45.8s
distributed.scheduler - INFO - Register worker <Worker 'tls://129.93.182.104:32827', name: htcondor--3436322.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tls://129.93.182.104:32827
distributed.core - INFO - Starting established connection
[                                        ] | 0% Completed | 47.1s
distributed.scheduler - INFO - Register worker <Worker 'tls://129.93.183.149:32837', name: htcondor--3436323.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tls://129.93.183.149:32837
distributed.core - INFO - Starting established connection
[                                        ] | 0% Completed | 50.0s
distributed.scheduler - INFO - Register worker <Worker 'tls://129.93.182.146:32816', name: htcondor--3436326.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tls://129.93.182.146:32816
distributed.core - INFO - Starting established connection
[                                        ] | 0% Completed | 50.5s
distributed.scheduler - INFO - Register worker <Worker 'tls://129.93.183.194:32961', name: htcondor--3436324.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tls://129.93.183.194:32961
distributed.core - INFO - Starting established connection
[#################                        ] | 43% Completed | 35.7ss
distributed.scheduler - INFO - Register worker <Worker 'tls://129.93.183.149:32838', name: htcondor--3436327.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tls://129.93.183.149:32838
distributed.core - INFO - Starting established connection
[##################################     ] | 91% Completed |  1min  1.7s
distributed.scheduler - INFO - Remove worker <Worker 'tls://129.93.183.149:32837', name: htcondor--3436323.0--, memory: 117, processing: 4>
distributed.core - INFO - Removing comms to tls://129.93.183.149:32837
distributed.batched - INFO - Batched Comm Closed: in <closed TLS>: ConnectionResetError: [Errno 104] Connection reset by peer
[#######################################  ] | 95% Completed |  1min 30.2s
distributed.scheduler - INFO - Register worker <Worker 'tls://129.93.182.104:32828', name: htcondor--3436328.0--, memory: 0, processing: 0>
distributed.scheduler - INFO - Starting worker compute stream, tls://129.93.182.104:32828
distributed.core - INFO - Starting established connection
[#######################################  ] | 95% Completed |  1min 33.8s
distributed.scheduler - INFO - Remove worker <Worker 'tls://129.93.183.194:32961', name: htcondor--3436324.0--, memory: 57, processing: 8>
distributed.core - INFO - Removing comms to tls://129.93.183.194:32961
[########################################] | 100% Completed |  1min 40.9s
```

[12]:
```
# Generates a 1D histogram from the data output to the 'MET' key. fill_opts are optional, to fill the graph (default is a line).
hist.plot1d(output['MET'], overlay='dataset', fill_opts={'edgecolor': (0,0,0,0.3), 'alpha': 0.8})
```

```
/opt/conda/lib/python3.7/site-packages/mplhep/_deprecate.py:56: DeprecationWarning: kwarg "densitymode" in function ``histplot`` is deprecated and may be removed in future versions: "unit"mode is not useful
  return func(*args, **kwargs)
```

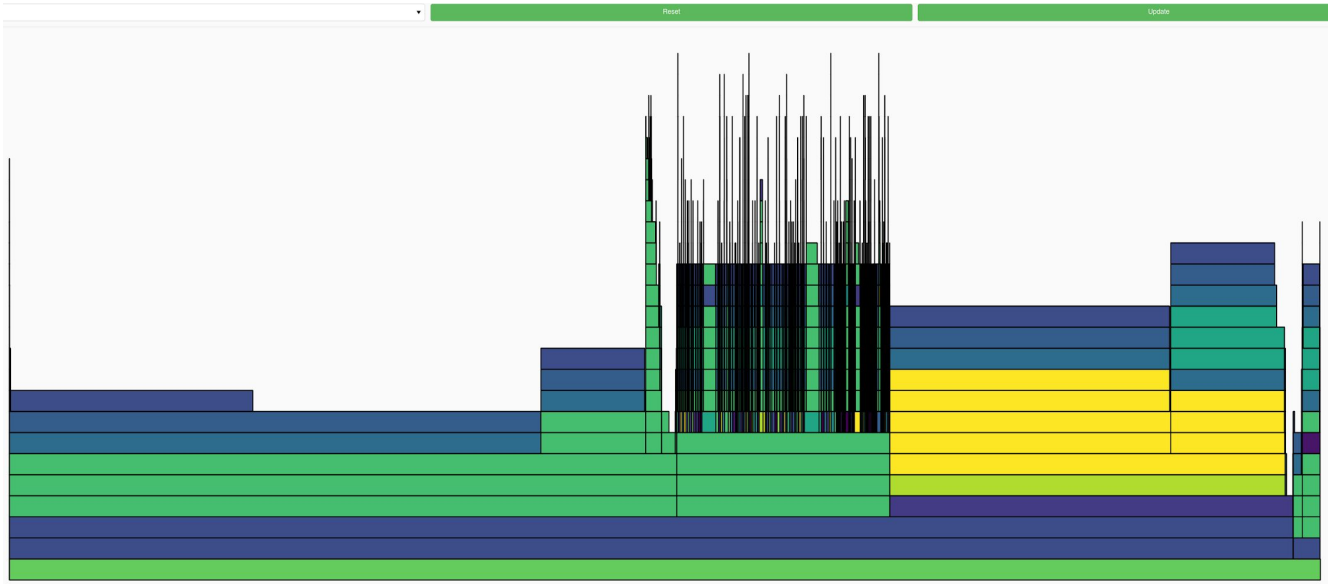
[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5c53acc10>
```



[13]:
```
# Easy way to print all cutflow dict values. Can just do print(output['cutflow']["KEY_NAME"]) for one.
for key, value in output['cutflow'].items():
    print(key, value)
```

```
all events 53446198
number of chunks 534
```

Status
Workers
Tasks
System
Profile
Graph
Info
Bytes stored: 2.51 GB
Task Stream
1/01
15s
30s
45s
:01:00
Tasks Processing
0
5
10
15
Progress -- total: 565, in-memory: 372, processing: 63, waiting: 10, erred: 0
METProcessor
485 / 534
reduce
6 / 30
bytes
1 / 1

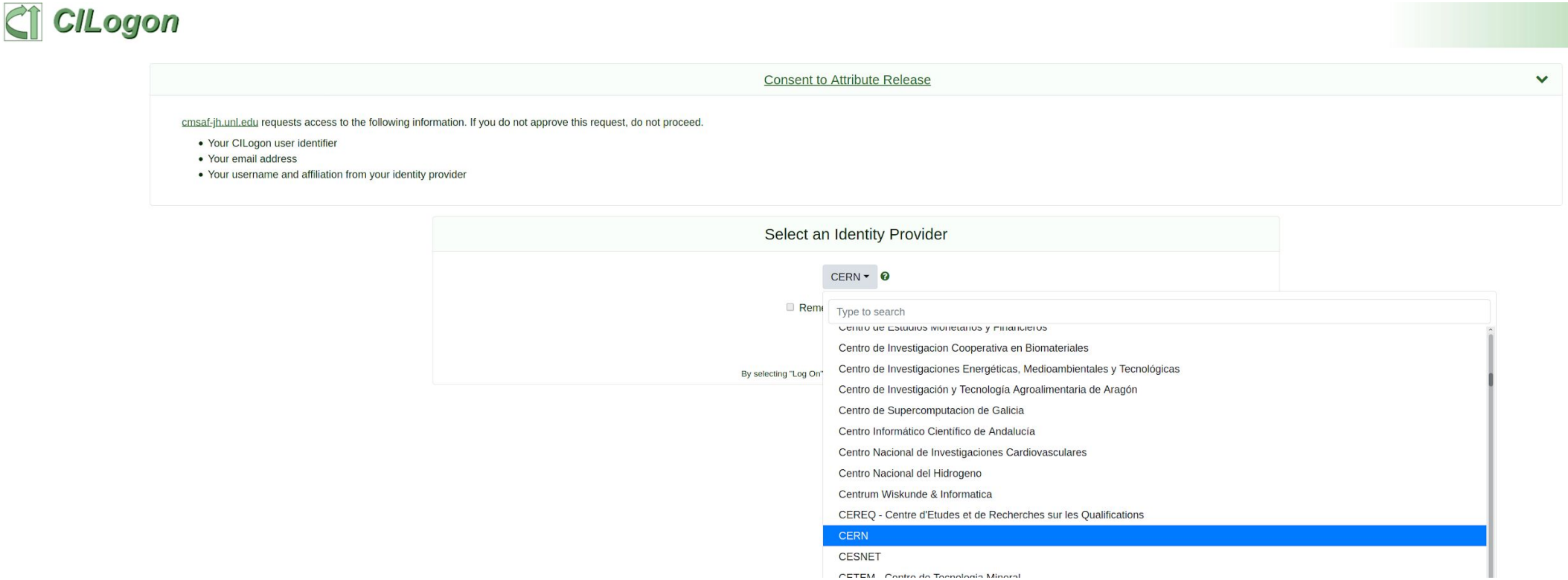

Reset
Update

Conclusions

- Still work in progress, a lot of ongoing improvements
- Integrating the native Dask scheduler with HTCondor jobs make scaling trivial
- Lots of scaling tests needed, both in terms of the jobs and the user base
- Deployment needs cleaned up and site specific customizations need removed and made more maintainable

Backup

CILogon



We will likely replace this with CMS Auth.

http://oauth.web.cern.ch/

Hardware Dedicated to Prototype (Kubernetes Cluster)

Role	Description	CPU	RAM
masters	2 x VMs living on old Dell machines	2	8GB
workers	4 x Dell R710s with disks for Rook.io	24	96GB
workers	3 x Sun X2200 (12 years old)	8	32GB
workers	5 x Sun X2200 (12 years old)	8	24GB
workers	2 x 4-in-2 Supermicro	16	64GB
workers	1 x 1U Supermicro	8	32GB

CMSAF @ UNL secrets

- All secrets are available in the directory */etc/cmsaf-secrets* at container startup *(but doesn't exist at build time)*
- The *BEARER_TOKEN_FILE* environment variable is going to be set to */etc/cmsaf-secrets/bearer_token*, matching what's expected in the XrdCl plugin.
- */etc/cmsaf-secrets/condor_token* is a condor IDTOKEN useful for submitting to T3.
- */etc/cmsaf-secrets/ca.key* is a CA private key useful for Dask
- */etc/cmsaf-secrets/ca.pem* is a CA public key useful for Dask
- */etc/cmsaf-secrets/hostcert.pem* is a host certificate and private key useful for the Dask scheduler.
- */etc/cmsaf-secrets/usercert.pem* is a user certificate and private key useful for the Dask workers.

CMSAF @ UNL XCache setup

- https://github.com/bbockelm/xrdcl-authz-plugin

```
$ xrdcp -f
root://xcache//store/data/Run2017B/SingleElectron/MINIAOD/31Mar2018-v1/60000/9E0F8458-EA37-
E811-93F1-008CFAC919F0.root /dev/null
Looking for token in file /home/cse496/bbockelm/projects/xrdcl-authz-plugin/xcache_token
[3.65GB/3.65GB][100%][==================================================][934.5MB/s]
```

CMSAF @ UNL XCache stats

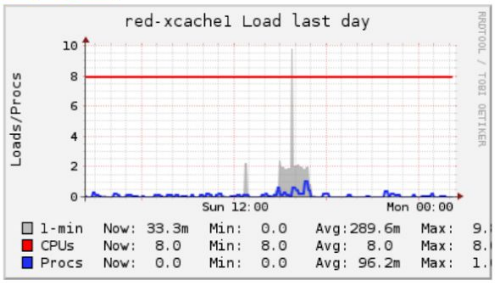
red-xcache1.unl.edu Host Report at Mon, 06 Jul 2020 02:16:55 -0500

Last | hour | 2hr | 4hr | day | week | month | year | job | or from _____ to _____ | Go | Clear

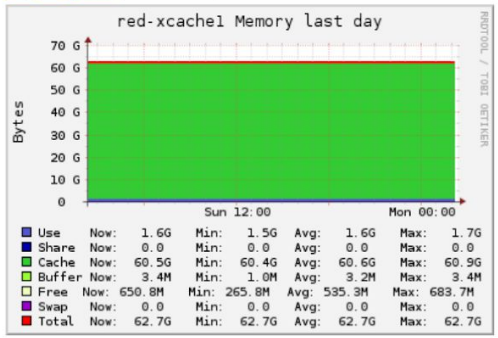
Holland Computing Center Grid > red-infrastructure > red-xcache1.unl.edu

Host Overview

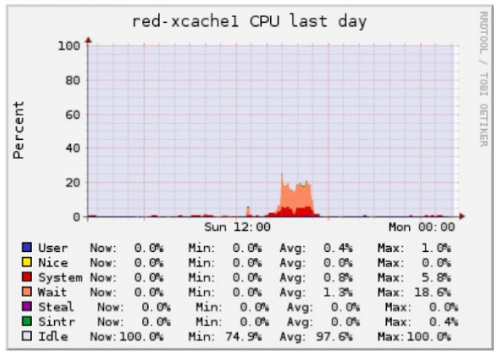
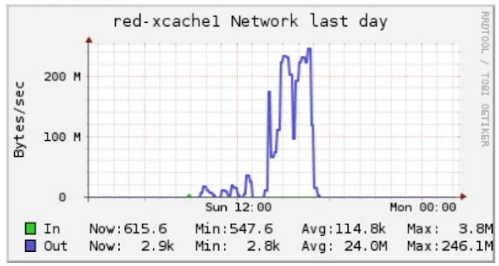
CSV JSON Inspect Hide/Show Events



CSV JSON Inspect Hide/Show Events



CSV JSON Inspect Hide/Show Events



CSV JSON Inspect Hide/Show Events