Caching setup at *PHYSnet* cluster & plans

FIDIUM Collaboration Meeting | RWTH Aachen | 30 September – 1 October 2024

Johannes Haller, Johannes Lange, Daniel Savoiu, Hartmut Stadie

U Hamburg commitments in FIDIUM

- Topic II – **Data lakes, distributed data, caching**
  - investigate and deploy data caching technologies
  - integrate dynamic data caches near newly integrated CPU resources

- Topic III – **Adaptation, testing, optimization**
  - deploy tools developed within FIDIUM to selected computing centers
  - integrate into production/analysis environments of HEP experiments
  - optimize to requirements for typical analysis workflows

***PHYSnet* cluster**

compute resources shared by all institutes of physics faculty

- heterogeneous cluster, various queues for diverse applications:
  - ***idefix.q*** – mixed single-threaded applications
  - ***infinix.q*** – for multi-node applications using MPI + InfiniBand
  - ***obelix.q***, ***epyx.q*** – for large-memory applications
  - ***graphix.q*** – for GPU applications

- parts reserved for exclusive use by various project groups
  - high flexibility for tailoring to individual/group use-cases
  - can integrate dedicated resources for HEP applications

- adaptable to HEP workflows using containerization technologies

up to
10 GE
uplink

~6 PB
hot storage

~675
compute
nodes

~14 400
CPU cores

~125 TB
RAM

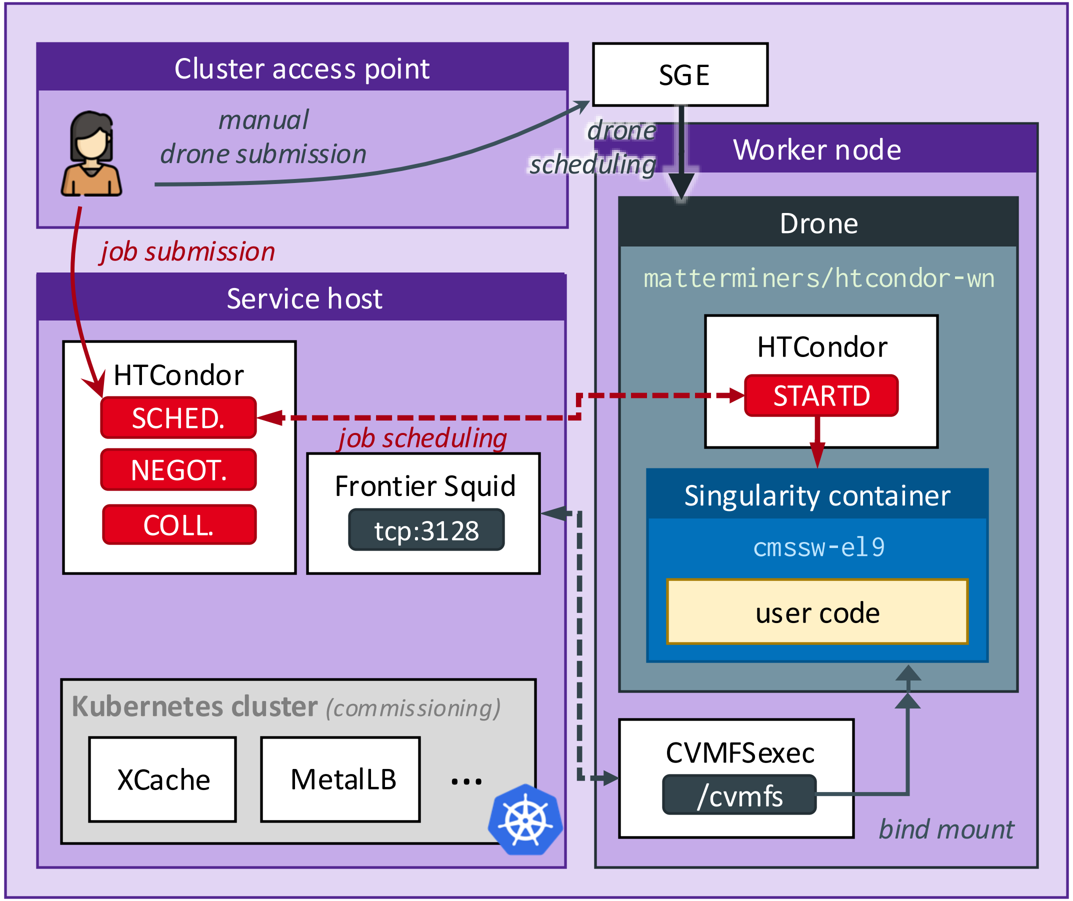
[Icons: flaticon.com]

	PHYSnet	**Typical WLCG sites / *NAF***
OS	***Ubuntu***	***RedHat*-based (SLC/CentOS)**
Batch system	***SGE****	***HTCondor***

) transition to **SLURM in progress*

Current setup

- working setup for scheduling HEP analysis jobs to PHYSnet cluster
  - central ***HTCondor*** instance
  - jobs scheduled to drone containers provisioned via native ***SGE*** batch system
- unpacked container images taken from `/cvmfs/unpacked.cern.ch`
- obtained dedicated resources for hosting HEP-specific services
- moved to EL9 grid environment for job containers



Caching for HEP workflows

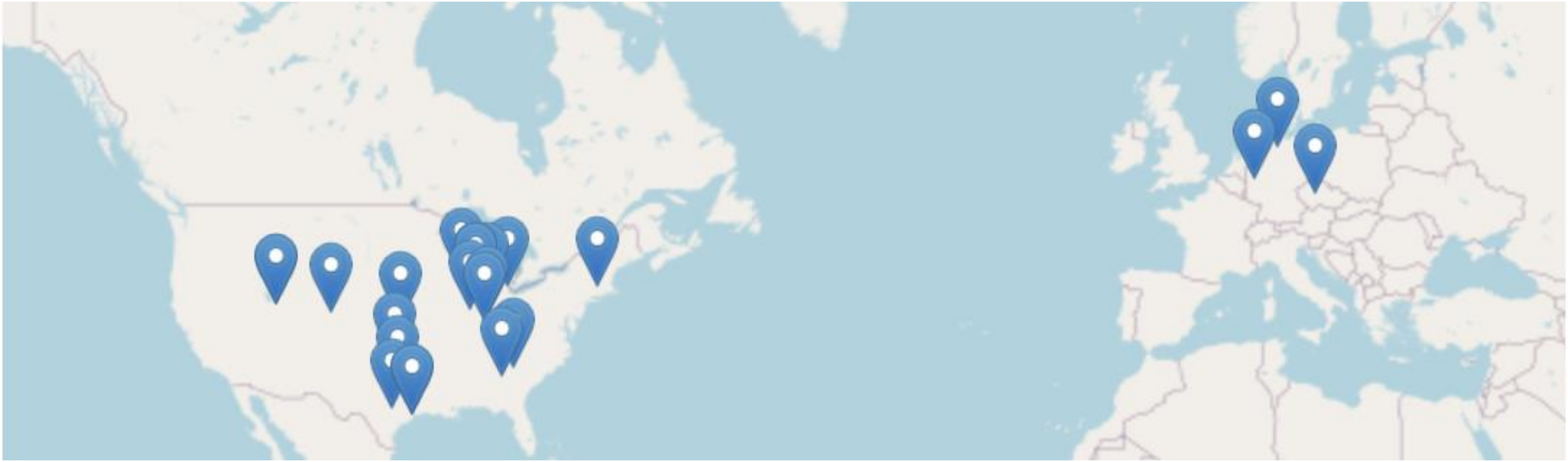
- HEP analysis workflows typically require a large amonunt of data from WLCG storage
  - large latency from WAN reads, read same files multiple times -> site-local caching solutions
- several broad strategies exist, including
  - **application-layer** → caching handled by I/O application, i.e. ROOT
  - **storage-layer** → caching/prefetching delegated to storage system
  - **lazy-download** → download remote files to local shadow copy
  - **disk-based proxy cache** → intercept WAN read requests, download & serve from local

supported by CMSSW

- ***XCache***: proxy caching for data access via ***XRootD*** protocol

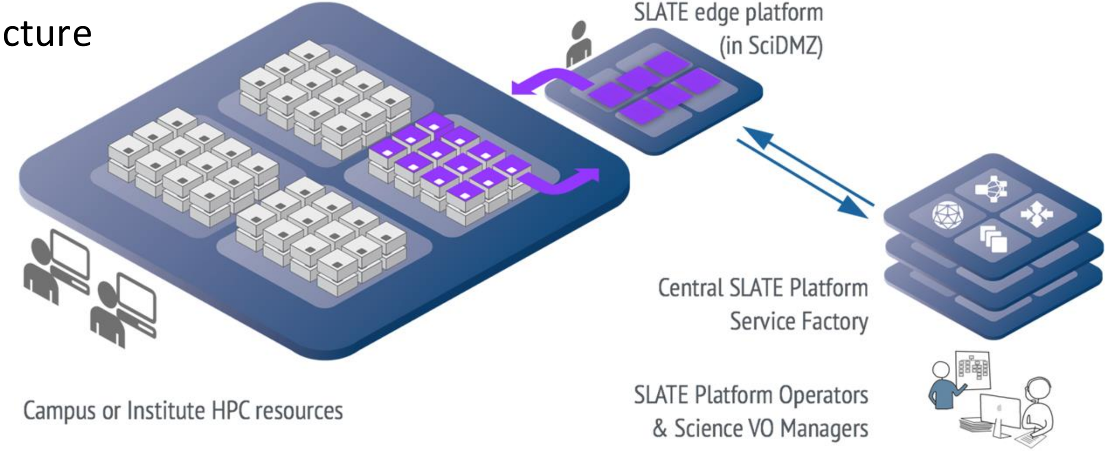
XCache deployment via *SLATE*

- recommended way to deploy ***XCache*** is via centralized provider ***SLATE*** *(Service Layer At The Edge)*
  - provide secure deployment of applications to remote sites via ***Kubernetes***
  - mostly US-based sites, some European sites also registered (Prague, Wuppertal, Hamburg)



XCache deployment via *SLATE*

- recommended way to deploy ***XCache*** is via centralized provider ***SLATE*** *(Service Layer At The Edge)*
  - provide secure deployment of applications to remote sites via ***Kubernetes***
  - mostly US-based sites, some European sites also registered (Prague, Wuppertal, Hamburg)
- set up ***Kubernetes*** cluster at PHYSnet, registered with SLATE federation
  - interaction between SLATE infrastructure and site via open-source client
  - installation not straightforward, some issues requiring client code modification
- deployment of ***XCache*** application to cluster to be tested

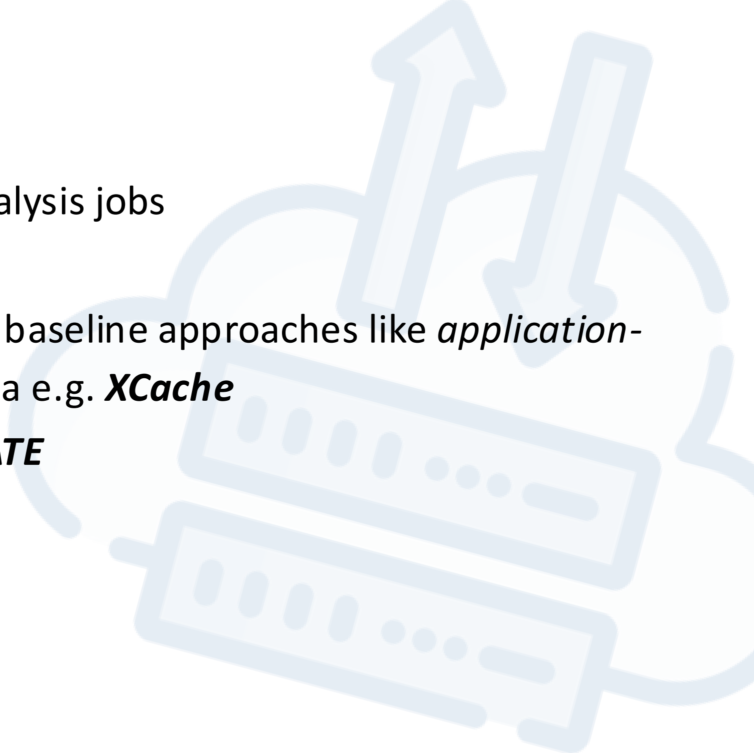


SLATE deployment details

- baseline site configuration required before applications can be deployed
  - initialize ***Kubernetes*** cluster using ***kubeadm***
  - add networking plugin for pod connectivity (***Calico*** recommended default, but heavy → chose more lightweight ***Flannel*** instead)
  - add load balancer (***MetalLB*** seems to be the only one supported and is required)
- register with SLATE (authentication via ***X.509***) to obtain personal access token for client
- install and run SLATE remote client to create and register cluster
  - client will install additional services/utilities to allow SLATE to operate with reduced privileges in ***Kubernetes*** cluster
- running into issues with client (***MetalLB*** not recognized, ingress controller IP detection fails, …)
  - partially fixed by editing client source code, looking for more stable solution

Summary

- continued development of ***PHYSnet*** cluster setup for HEP analysis jobs
  - obtained dedicated resources for testing
- investigated several options for caching input data, including baseline approaches like *application-layer/lazy-download* and disk-based ***XRootD*** proxy caching via e.g. ***XCache***
- preparations for ***XCache*** deployment via ***Kubernetes*** and ***SLATE***
  - mostly done but some issues with SLATE client to be fixed



Next steps

- fix issues and finalize ***XCache*** deployment
- evaluate performance of caching with the ***XRootD*** proxy approach
- possible option to set up a satellite ***dCache*** instance at PHYSnet

Thank you for your attention!