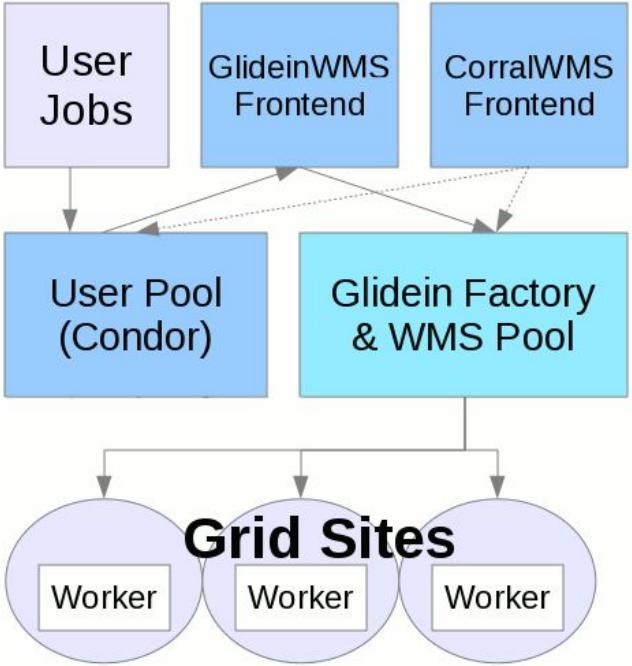
Experiences integrating FZJ HPC Resources into the CMS Global Pool

Thomas Madlener

3rd TA Mini Workshop | Apr 09 2024

Global Pool

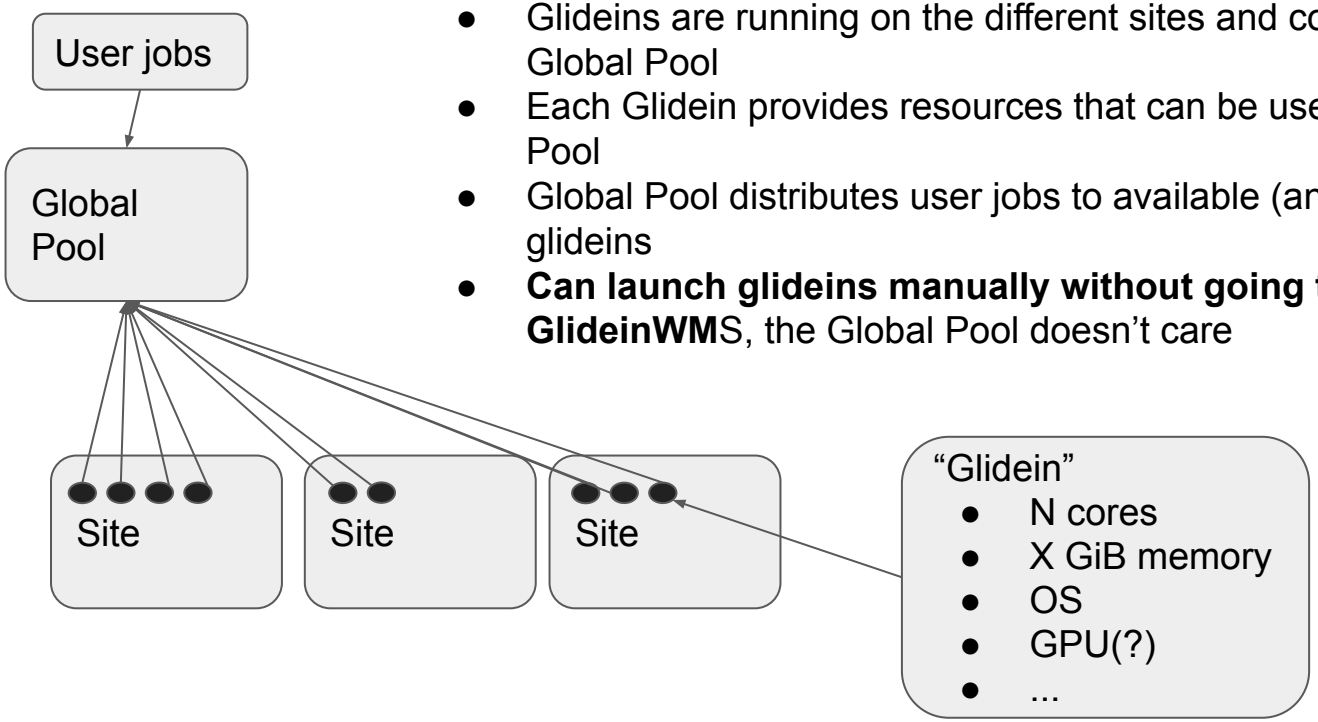
- User job requests are sent to the “User Pool” (HTCondor)
- The GlideinWMS Frontend polls the user pool and ensures that enough resources are available
  - Resources are so called “glideins” in this case
  - Submits requests to to Glidein factory
- Glidein Factory and WMS Pool receive requests and submits HTCondor startd wrappers (glideins) to computing sites
- Site receives glidein jobs and start a HTCondor startd that joins the User Pool and becomes available as resource
- User jobs are matched to these resources
- Users only see the User Pool that “magically” grows/shrinks to match demand



https://glideinwms.fnal.gov/doc.prd/index.html (has nice animation of process)
https://twiki.cern.ch/twiki/pub/LCG/DiscussionsOnCMSSpecificCRIC/GlideinWMS-Factory-CERN.pdf

Global Pool and “manual glideins”

- User jobs go to the “Global Pool”
- Glideins are running on the different sites and connect to the Global Pool
- Each Glidein provides resources that can be used by the Global Pool
- Global Pool distributes user jobs to available (and matching) glideins
- **Can launch glideins manually without going through GlideinWM**S, the Global Pool doesn’t care

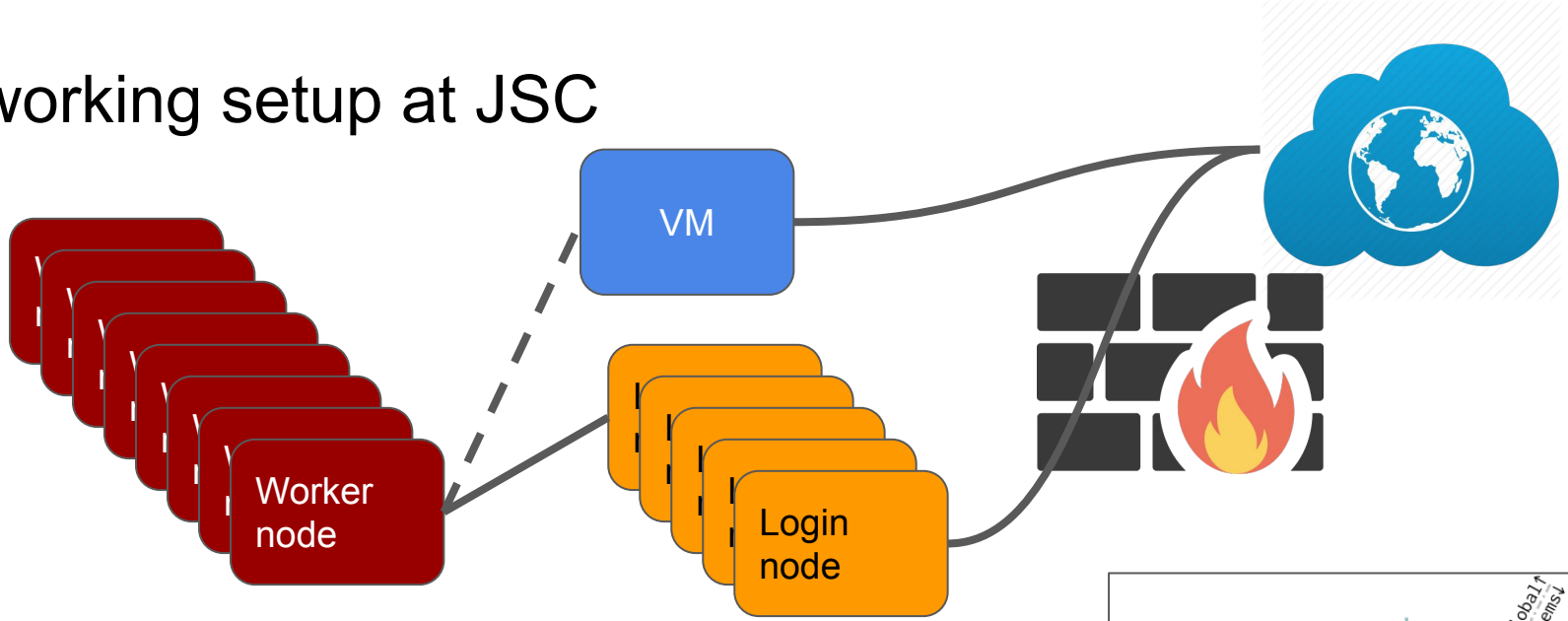


Requirements for running CMS jobs

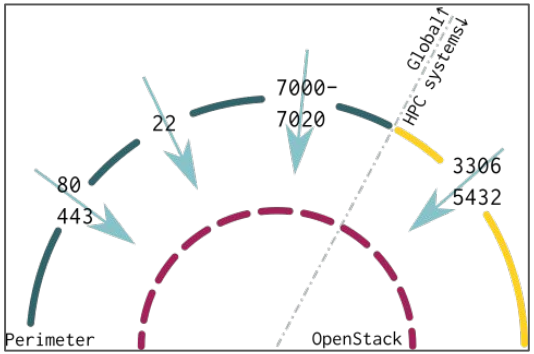
- Hard- and software environment can run CMS software
- Glidein factory can get pilot jobs (glideins) onto the resource
- Glidein is able to report back and get work (i.e. outgoing network connection)
- Access to resource specific settings
- Access to CMS software and conditions data
- Potentially access to input data (depending on job type)
- Access to storage to write output data
    - Ideally also directly able to report to data CMS data management tools (RUCIO)

D. Hufnagel, Enabling opportunistic resources for CMS Computing Operations (CHEP 2015)
https://inspirehep.net/literature/1413191

Networking setup at JSC



- Worker nodes can only connect to Login nodes
- Login nodes have outbound connections
  - But only on ports 80 (http) and 443 (https)
  - We need other ports as well
- VMs in DMZ with fewer network restrictions
  - Port 5432 and 3306 reachable from DMZ



https://www.fz-juelich.de/en/ias/jsc/systems/scientific-clouds/hdf-cloud/network-setup

Conditions data

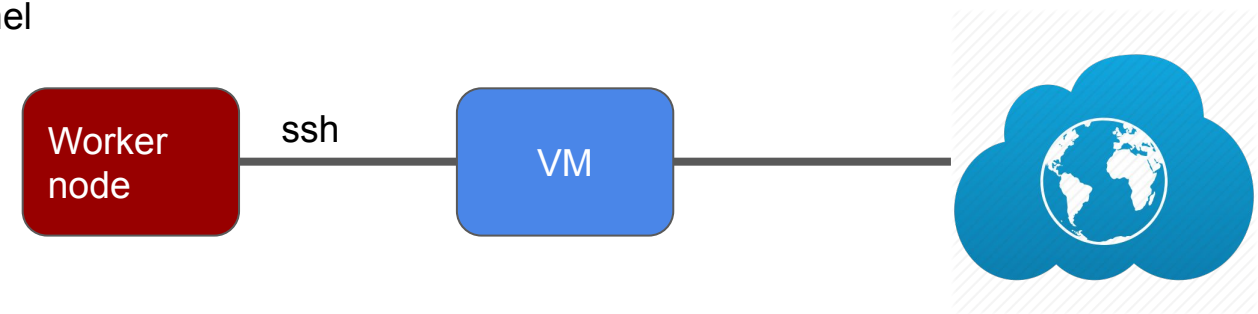
- Accessed via Frontier distributed database caching system
  - Uses the squid caching tool with some additional patches
- http-based protocol and RESTful API
- Need to be able to run a squid proxy and connect to it from worker nodes ✅
  - Can start squid proxy on VM and redirect http requests through it from worker nodes (**via one of the open ports**)
  - Can reach the central Frontier database
- Squid proxy usually run on separate node serving multiple worker nodes
  - Needs O(100 GB) disk space for caching and log files

Software environment (and situation at JSC)

- CMS software is distributed via CVMFS
- Using apptainer container to run jobs (depending on hardware and OS requirements)
  - Unpacked container images for different OS and hardware types are also distributed via CVMFS
- Need to be able to mount CVMFS on JSC ✅
  - Possible via cvmfsexec and bind-mounting into the container
  - On typical HEP sites the necessary repositories are already mounted
  - Network traffic via squid running on VM
- Need to be able to run apptainer images
  - Possible to pull images from dockerhub and then run
  - Not possible to run unpacked images from cvmfs directly (probably apptainer setting)
  - **Not possible to run nested containers**

Communication with Global Pool

- Glideins need to be able to connect to the Global Pool from worker nodes
- On other HPC sites, e.g. via ssh tunnel to login node and outbound connection from there
  - Not enough network connectivity on JSC login nodes
- Can use VM to reach outside network via proxy setup
  - ssh tunnel from worker to login node (**using the second open port**)
  - Use proxychains to route all network traffic (except cvmfs & conditions data) through ssh tunnel



Putting everything into a batch job

- Setup ssh tunnel
- Mount cvmfs repositories (using cvmfsexec)
- Launch apptainer container and bind-mount cvmfs repositories to /cvmfs

"Setup" on worker node

- Enter the now running container

- Pre-load proxychains library (LD_PRELOAD)
- Launch glidein
- Enjoy

Work that is done inside the container that is launched in the setup

Automating glidein submission

- Using COBalD / TARDIS
  - Developed at KIT
  - Used for transparent integration of other HPC resources as an extension to T1_DE_KIT
- Monitors usage of glidein for given site(s)
- Dynamically adds / removes glideins depending on usage
  - Slightly different approach than usual where “demand” is used
- Run on VM and submit (slurm) batch jobs via ssh

COBalD – the Opportunistic Balancing Daemon

docs passing | build failing | codecov 0% | pypi v0.12.3 | license MIT | DOI 10.5281/zenodo.5681401

The `cobald` is a lightweight framework to balance opportunistic resources: cloud bursting, container orchestration, allocation scaling and more. Its lightweight model for resources and their composition makes it easy to integrate custom resources and manage them at a large scale. The idea is as simple as it gets:

Start good things.
Stop bad things.

TARDIS - The Transparent Adaptive Resource Dynamic Integration System

Welcome to the TARDIS documentation!

The `TARDIS` resource manager enables the dynamic integration of resources provided by different resource providers into one overlaybatch system. `TARDIS` relies on COBalD - the Opportunistic Balancing Daemon in order to balance opportunistic resources. Hence, `TARDIS` is implemented as a `COBalD` service.

Installation

```
python3 -m pip install coba
```

Configuration of C

https://gitlab.desy.de/thomas.madlener/cms-drp-jsc

Summary

- Pool knows about Glideins that are registered with it
  - Can send payloads to these glideins if requirements of jobs match what it offers
  - Monitors these glideins and keeps track of which glideins are working on which payload
  - Glideins are not tied to user jobs, they just offer resources for jobs to run in
- Pool is agnostic to how these glideins are launched
  - GlideinWMS responsible for providing enough glideins for usual grid workflows
  - **JSC: manually submitting glidein batch jobs to the (host, aka slurm) batch system**
- (Ab)using some resources available to us in JSC DMZ we can get simulation jobs to run
  - No input data required
  - Output data handled via the usual CMS stageout tools and tunnel / proxy setup
  - Currently still very targeted job submission
- Glidein submission automated via Cobald / Tardis
- Prototype setup working -> some tweaks necessary for production

Open points / observations

- Currently running a slightly non-standard glidedin wrt CMS
  - Standard glidein requires possibility of running nested containers
  - Some repercussions for production usage (e.g. cannot choose OS flexibly)
- “Proper scale testing”
  - Regularly run 256 jobs (1 node on Jureca), filling all slots of a glidein
  - Using parallel 2 glideins (512 jobs) we observed some network issue(?) in the past, but not reproducible at the moment (no changes to our setup)
- “Network issue”:
  - Glidein lost connection with pool -> Cobald / Tardis could no longer see it and killed it on the host batch system
  - Keeping glidein batch job alive -> Payloads complete normally and glidein at some point reconnects with pool
- Full production scale will require some interaction with JSC admins

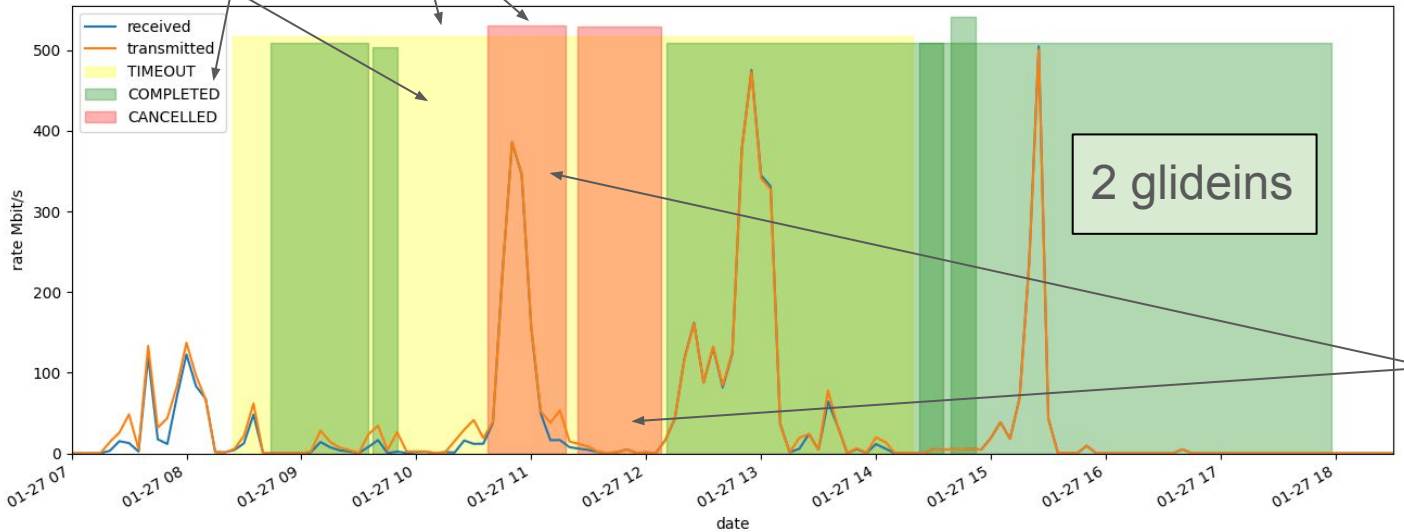
Backup

Network usage on VM

5 min averages

Height is random!
(visualization only)

Slurm exit status



"Idle" (no payloads), Cobald / Tardis keeps one glidein always

Excluded network overload as reason for glidein disconnects

Data access

- Data usually stored/replicated on different sites
- Accessing via network using the site that is “closest”
- Users can black/whitelist certain sites in their configuration
- Replication and registration in CMS central database via RUCIO
- To access data need necessary authentication via grid proxy and outbound connection
  - With proxy setup can copy data to worker nodes via xrdcp
  - Automatic stage-out via T1_DE_KIT to T2_DE_DESY is working for simulation jobs (no inputs)
- Not completely solved yet!

Timeline of User Job

User submits job to User Pool

Glidein WMS Frontend submits request for glideins (if not enough resources available)

Glidein Factory receives requests from Frontend

Submits glideins as jobs to host batch system

glidein (pilot) starts on host batch system

Sets up environment and runs validation

Reports back to User Pool and becomes available resource

User Pool matches glideins (resources) to the user jobs and sends them to the worker nodes

User jobs are started on the worker nodes that are “reserved” by the glidein

User job runs; accessing conditions data, input data, etc. and producing output data

User job finishes

Output data is handled by “Stage Out” procedure that registers data in central database and handles moving it appropriately

Glidein remains available and if new user jobs arrive they can be matched and run in the same resource

Allocation for the glidein on host batch system ends and resources are returned to the host

Global Pool management

Usage of site ressources

Worldwide LHC Computing Grid (WLCG)

- Tier-0 @CERN
  - Prompt reconstruction, long-term data storage
- Tier-1
  - Long-term storage (partial data)
  - Reprocessing
  - Distribution to Tier-2
- Tier-2
  - Analysis and Simulation tasks
  - E.g. @DESY
  - Some storage
- Tier-3
  - Smaller resources without formal WLCG agreement
- Challenge: Integrate different sites transparently for users