Usage of HTCondor at SubMIT

Jan Eysermans on behalf of the SubMIT team
SubMIT Workshop
January 23 2026

Condor as Workflow Management System

HTCondor is a system that runs large numbers of independent jobs efficiently on shared computing resources

Why we use HTCondor on SubMIT

- HTCondor is made to run **many independent jobs**
- You **submit once**, the system runs jobs **when resources are free**
- Jobs don’t need to talk to each other
- Failures are handled automatically (retry, reschedule)
- *Best choice when your work scales by the **number of jobs***, e.g. simulations, parameter scans, large analysis campaigns

HTCondor maximizes throughput and fairness for shared resources, whereas Slurm is for tightly coupled, high-performance jobs

Condor vs. Slurm

Many resources locally but also world-wide

Jobs run in an isolated environment

- No direct access to subMIT environment
- Non-uniform resources (different OS)
- Need to port your environment to the worker node
- Overcome using singularity images

Condor will schedule the jobs automatically and enforce fair sharing across users and groups

Local resources restricted to subMIT

Jobs run within the subMIT environment

- Easy access to subMIT shared file system (home/work/ceph) → can directly read/write
- Easy input/output handling
- Uniform resources (same OS, libraries…)

Slurm will also schedule the jobs automatically based on recent usage, job size, etc.

A simple Example

To submit a job to condor, you need two files:

- A submission file steering the submission (what resources, types, …) → more on that later
- A wrapper (bash) script that contains all the work to be done

submission.sub

```
request_disk    = 1024
executable      = script.sh
arguments       = $(ProcId)
output          = $(ClusterId).$(ProcId).out
error           = $(ClusterId).$(ProcId).err
log             = $(ClusterId).$(ProcId).log
+DESIRED_Sites = "mit_tier3"
arguments       = "$(ProcId)"
queue 10
```

script.sh

```
#!/bin/bash

echo "I am a HTCondor job!"
echo "I have landed in $(hostname)"
echo "I have received parameter $1"
echo "That's all!"
```

To submit a job, do: `condor_submit submission.sub`

- Condor will give you a unique ID for that submission, called *ClusterId*
- The *ProcID* contains the different processes inside a job and are the actual jobs (*ProcID* starts from 0)
- In this case individual 10 jobs will run in a single cluster, all executing `script.sh`

Tracking a Submitted Job

To check the status of my jobs, use: `condor_q`

- Initially the jobs stay in IDLE till the right resources are found

3 files per procID are typically generated when a job is running and/or finished

- Output file: the output of the wrapper script (script.sh in this example)
- Error file: anything written to the error stream will end up here
- Log file: an internal Condor log file, very useful if jobs fail or do not run

There can be many reasons why a job is executed but fails

- Sometimes jobs are stuck, don't run or land on a bad worker node
- Resources are not correctly matched (see later)
- Transfer issues

It is advised to inspect the output files to further debug and find the cause

- In case of occasional transient issues, it is possible to configure your submission to let condor retry the job automatically

```
max_retries = 3
```

Custom Job Arguments

Suppose I want to run some jobs with a given input argument (e.g. the seed of an event generator)

- The seed number should be transferred to the job and executable
- The program generates an output file that should be transferred back to subMIT

An example is given below, where we introduce a unique parameter *SEED* that can be used throughout the environment

- Can also be loaded from a text file

submission.sub

```
request_disk    = 1024
executable      = script.sh
arguments       = $(SEED)
output          = $(ClusterId).$(ProcId).out
error           = $(ClusterId).$(ProcId).err
log             = $(ClusterId).$(ProcId).log
+DESIRED_Sites = "mit_tier3"
arguments       = "$(ProcId)"
transfer_output_files = output_$(SEED).txt
queue SEED in (288575 478994 706524 229149 438593)
## or queue arg from inputs.txt
```

script.sh

```
#!/bin/bash

echo "I am a HTCondor job!"
echo "I have landed in $(hostname)"
echo "I have received parameter $1"
echo "That's all!"
echo "output" > output_$1.txt
```

What Resources are Available?

SubMIT has access to many external resources

- On campus: T3 (1000 cores)
- OSG pool: dozens of campus, national labs, and non-profit organizations (OSG = Open Science Grid)
- CMS global pool: CMS sites worldwide (need to be a CMS member) – including on-campus T2 (>10000 cores)

Given the large available resources, could reach 10k+ jobs running at once

- Many more resources available than Slurm

How to specify in my submission script where a job should go?

```
# On campus T3/T2 resources
+DESIRED_Sites = "mit_tier2,mit_tier3"
Requirements   = (BOSCOCluster=!="t3serv008.mit.edu" && BOSCOCluster=!="ce03.cmsaf.mit.edu" && BOSCOCluster=!="eofe8.mit.edu")

# OSG
+ProjectName    = "MIT_submit"
Requirements   = (OSGVO_OS_STRING=="RHEL 9")

# CMS pool
+AccountingGroup = "analysis.myusername"
+DESIRED_Sites   = "T2_AT_Vienna,T2_BE_IIHE,T2_BE_UCL,T2_BR_SPRACE,..."
```

Transferring Input and Output Files

HTCondor jobs are remotely executed, you don't have access to /home /work /ceph etc.

- You need to transfer whatever you need as input to the remote machine your job is running to
- This is different than Slurm where the SubMIT environment is available

Via submission script

- You can add the following to your submission script

```
transfer_input_files     = <your comma-separated list of files>
transfer_output_files    = <your comma-separated list of files>
```

- This will transfer your files to the compute node
- Limited to ~ 250MB!

Via XRootD – for larger files

- XRootD is a protocol to transfer files, allows you to transfers files remotely
- You can read/write data remotely in your executable (bash) script

```
xrdcp root://submit50.mit.edu//data/user/j/jaeyserm/input.txt .
xrdcp output.txt root://submit50.mit.edu//data/user/j/jaeyserm/output.txt .
```

- But you need to set up certificates to authenticate yourself to the network [NEED TO ADD INFO]

Environment: Singularity and CVMFS

As HTCondor jobs are executed on remote machines (worker nodes), the environment is usually different than what you have on SubMIT

- On SubMIT you compile/test/run your program with custom libraries: “it works on my machine”
- Running on Condor can fail due to environment and library mismatch
- Further complication by the fact the OS version of the worker nodes can be different (currently a mix of CC7/RHEL8/ALMA9)

Two ways to solve such issues

Via transfer

If you don’t need a lot of software, and you can package it (perhaps by compiling it in a way that is self-contained), you can transfer it via the methods outlines in the previous section: either through the submission script or HTCondor

Via CVMFS – a shared read-only filesystem that contains many libraries and singularity images

CVMFS is mounted on subMIT and all clusters connected to subMIT via HTCondor, and supports the distribution of containers. In order to use a container in your jobs, you can specify which image you want via, e.g

```
+SingularityImage= "/cvmfs/singularity.opensciencegrid.org/opensciencegrid/osgvo-el9:latest"
```

You can even distribute your own containers to CVMFS, or transfer it manually

Matching Requirements

Condor is a (fair) shared-based system

However, it is important to adjust the submission to the resources you need:

- Memory allocation per job: “`request_memory = 1024`” (MB) → *the most important one!*
- CPUs per job: “`request_cpus = 1`”
- Disk space per job: “`request_disk = 1024*500`” (kB)
- Time? Not considered mainly? (at CERN it is)

The more you match your jobs to their requirements, the better the system can balance and distribute the load among the resources and other users

- Excessive unused/misconfigured resources is monitored and will be reported

In case of extraordinary needs (e.g. space/memory/number of jobs), please contact us!

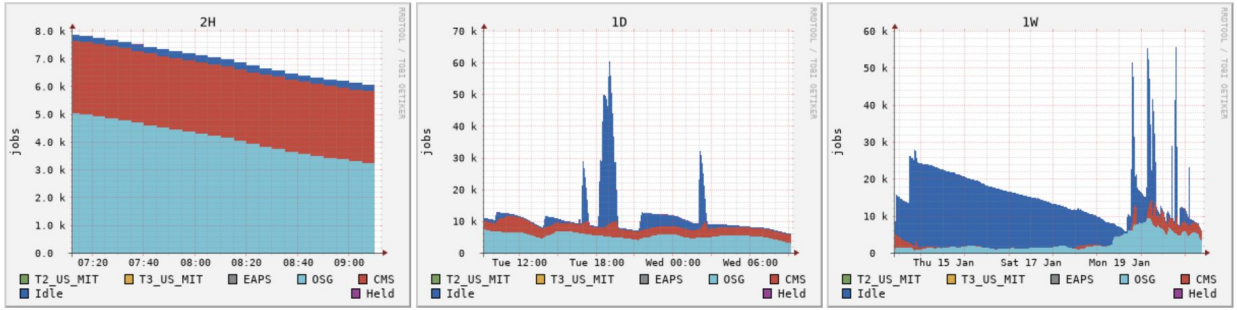
Job Monitoring

All active jobs can be monitored on the terminal using `condor_q`

All non-active (finished/removed) jobs can be traced back using `condor_history`

Time-dependent monitoring can be found here: https://submit.mit.edu → "Status condor queue"

- Contains general overview as well as user specific



User	Idle	Held	Running	MIT			OSG	CMS	Total
				T2_US_MIT	T3_US_MIT	EAPS			
jaeyserm	11	0	2588	0	0	0	0	2588	2599
paus	0	0	0	0	0	0	0	0	0
sahughes	5	0	3262	0	0	4	3258	0	3267
dr_orona	202	0	0	0	0	0	0	0	202
emoreno	12	0	0	0	0	0	0	0	12
Total	230	0	5850	0	0	4	3258	2588	6080

Conclusions

HTCondor as Workflow Management System embedded and heavily used at SubMIT

- Many on-campus and global (CPU) resources available

Extended user guide and tutorials available

- https://submit.mit.edu/submit-users-guide/running.html#htcondor
- https://github.com/mit-submit/submit-examples/tree/main/htcondor

If you encounter any problems, features you want to use, or discussion on how to integrate the usage of Condor in your workflow, please let us know: submit-help@mit.edu

- Your experience is valuable to us!