HEP User Training Workshop

Cluster information

The cluster, placed in academia sinica, is registered in Worldwide LHC Computing Grid (WLCG) as Tiar3 in Taiwan, called `T3_TW_TIDC` and provides resources of 768 CPU cores and 500+ TB storage. For local users, the cluster also donates a local job submission UI called `TIDC-ui01` under CentOS7 (maybe upgrade in future?). To be able be access this cluster, you have to apply a personal account in here. Note that people affiliated to NTU and NCU should fill Kai-Feng Chen and Chia-Ming Kuo as PI, respectively.

Once your application is approved, you can access the cluster by using SSH

```
ssh -YC <your account name>@tidc-ui01.grid.sinica.edu.tw
```

and your screen output should look like

You can also directly access files stored in `T3_TW_TIDC` (`/eos/cms/store/user/`) if you submit CRAB jobs and point the storage site to this T3.

CMS Virtual Organization Membership (CMS VO)

Besides `T3_TW_TIDC`, if your analysis needs to access files in the other Tiar clusters of WLCG, `tidc-ui01` also is available to create your own Grid certificate for access under the premise that you have a cern account and CMS VO Membership.

To have VO Membership, please follow the below steps :

- Step 1: Create a personal certificate file in here and click "New Grid User certificate". Follow the indication in this web and you can download a certificate file called `myCertificate.p12` . Note that the duration of the certificate is only 1 year not permanent.
- Step 2: Include the certificate `myCertificate.p12` to the web browser like chrome, firefox in order to access VOMS Admin server here.

- Step 3: Fill some personal information and your cern account in VOMS Admin server, you can be a member of the CMS VO. More details in here.

Once you have the VO Membership, to get access to WLCG through `tidc-ui01`, please put your certificate `myCertificate.p12` to `tidc-ui01` and then do the following commands. These commands only need to do once unless you update your `myCertificate.p12` .

```
myCert="myCertificate.p12"

cd ~
mkdir .globus
cd ~/.globus
cp ~/${myCert} .

openssl pkcs12 -in ${myCert} -clcerts -nokeys -out usercert.pem
openssl pkcs12 -in ${myCert} -nocerts -out userkey.pem
chmod 400 userkey.pem
chmod 400 usercert.pem
cd ~
```

Finally, to authenticate the grid certificate, you have to do the following commands. A private key `x509up_u<your UID>` with a duration of 168 hours will be created and assigned to `X509_USER_PROXY` environment variable. Then you can use the grid resource according to `X509_USER_PROXY` as accessing the CMS data or simulated samples in DAS or submitting CRAB jobs. Note that you have to reassign `X509_USER_PROXY` whenever you login to `tidc-ui01` or redo the following commands again when `x509up_u<your UID>` is expired.

```
voms-proxy-init -voms cms -rfc -out ${HOME}/x509up_u${UID} --valid 168:00
export X509_USER_PROXY=${HOME}/x509up_u${UID}
```

CernVM File System (CVMFS)

For those who would like to use cms software (CMSSW), the cluster provides CernVM File System (CVMFS) service by

```
source /cvmfs/cms.cern.ch/cmsset_default.sh
```

If you usually need to use CMSSW, please add the above to `~/.bashrc` so that you don't need to do it whenever you login to `tidc-ui01`. Then, you can start to setup CMSSW by

```
export SCRAM_ARCH=slc7_amd64_gcc700

cmsrel CMSSW_10_6_38
cd CMSSW_10_6_38/src
cmsenv
git cms-init

git cms-addpkg ...
...
```

where `SCRAM_ARCH` points out which architecture you will use for CMSSW setup. For example, `slc7_amd64_gcc700` means the software is built under `CentOS7` OS + `AMD64` 64 bits instruction set architecture + `GCC7` . However, since the cluster is `CentOS7` , you can only specify `slc7_amd64_gccX` . If you want to check which CMSSW version is supported under a given architecture, you can use

```
scram list | grep "CMSSW"
```

to list as

```
CMSSW           CMSSW_10_6_32
                                --> /cvmfs/cms.cern.ch/slc7_amd64_gcc700/cms/cmssw/CMSSW_10_6_32
CMSSW           CMSSW_10_6_32_patch1
                                --> /cvmfs/cms.cern.ch/slc7_amd64_gcc700/cms/cmssw-patch/CMSSW_10_6_32_patch1
CMSSW           CMSSW_10_2_16_UL3
                                --> /cvmfs/cms.cern.ch/slc7_amd64_gcc700/cms/cmssw/CMSSW_10_2_16_UL3
CMSSW           CMSSW_10_6_33
                                --> /cvmfs/cms.cern.ch/slc7_amd64_gcc700/cms/cmssw/CMSSW_10_6_33
CMSSW           CMSSW_10_6_34
                                --> /cvmfs/cms.cern.ch/slc7_amd64_gcc700/cms/cmssw/CMSSW_10_6_34
CMSSW           CMSSW_10_6_35
                                --> /cvmfs/cms.cern.ch/slc7_amd64_gcc700/cms/cmssw/CMSSW_10_6_35
CMSSW           CMSSW_10_6_35_patch1
```

Miniconda (Recommendation for python users)

If your analysis framework is based on `python` as you need to install a lot of python modules i.e. `numpy` , `matplotlib`, `coffea`, …, instead of using `pip install` , it's recommended to use Miniconda or Miniforge3, these package managers for installation. The advantages of the package managers are

- Safely install python modules without affecting libraries in Linux.
- Provide independent environments to avoid interfering with Linux libraries and other environments.
- Automatically resolve the dependence of different python modules in order to avoid any module and version conflict when installing lots of python modules.

Setup

- Step 1 : Download Miniconda/Miniforge3 to `$HOME`

```
# Miniconda
wget "https://repo.anaconda.com/miniconda/Miniconda3-latest-$(uname)-$(uname -m).sh" # for Linux
wget "https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-arm64.sh" # for Mac

# Miniforge3
wget "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-$(uname)-$(uname -m).sh" # for Linux
wget "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-MacOSX-arm64.sh" # for Mac
```

- Step 2 : Install Miniconda/Miniforge3

```
# Miniconda
sh Miniconda3-latest-Linux-x86_64.sh # for Linux
sh Miniconda3-latest-MacOSX-arm64.sh # for Mac

# Miniforge3
bash Miniforge3-Linux-x86_64.sh # for Linux
bash Miniforge3-MacOSX-arm64.sh # for Mac
```

Suggestion : When going to the final step of the installation as the following, please click `no`

```
Do you wish the installer to initialize Miniforge3
by running conda init? [yes|no]
[no] >>> no
```

- Step 3 : Add the following to `~/.bashrc` to set conda environment

```
# Miniconda
source ${HOME}/miniconda3/etc/profile.d/conda.sh

# Miniforge3
source ${HOME}/miniforge3/etc/profile.d/conda.sh
```

Basic usage

Suppose you would like to create an environment named “myenv" with `python 3.9` and you can simply do the following command

```
# Miniconda
conda create --name myenv python=3.9

# Miniforge3
mamba create --name myenv python=3.9
```

You can enter this environment space by

```
# Miniconda
conda activate myenv

# Miniforge3
mamba activate myenv
```

or exist from it by

```
# Miniconda
conda deactivate

# Miniforge3
mamba deactivate
```

You can also install `numpy`, for example, into “myenv” when you are in “myenv” by

```
# Miniconda
conda install numpy

# Miniforge3
mamba install numpy
```

If you need to install a lot of python modules at the same time, you can create a `environment.yml` file as

```
name: myenv
channels:
  - conda-forge
  - defaults
dependencies:
  - python>=3.10
  - coffea
  - xrootd
  - numba
  - vector
  - dask
  - distributed
  - dask-jobqueue
```

```
  - pandas
  - matplotlib
  - xgboost
```

where

- `name` is the environment name;
- `channels` points out which projects the python modules originate from ; you can choose `default`, `conda-forge`, `bioconda`;
- `dependencies` records which python modules need to be installed in the environment.

Then install them by

```
# Miniconda
conda env create -f environment.yml

# Miniforge3 (recommended)
mamba env create -f environment.yml
```

Recommend to use `mamba` to accelerate and avoid failure when resolving packages if there are too many modules need to be installed.

HTCondor submission

In fact, `tidc-ui01` itself is only 4 CPU cores so that it's inefficient to execute very heavy programs in this UI. Instead, we should perform parallel computing under `tidc-ui01` through HTCondor job submission, that is, to use the 768 CPU cores in this T3 cluster. You can find out the HTCondor introduction in here.

Basic usage

To submit common HTCondor jobs, two files need to be prepared, an executable file and a HTCondor configuration file. The executable file is usually a bash script or a python script which is customized by users. However, for bash scripts, It's strongly recommended to write absolute paths and avoid some default environment variables/alias such as `$HOME`, to avoid any job failed because you never know where the jobs run. Here has two example scripts:

- Jobs under CMSSW

```
#!/bin/bash

# Setup grid certificate if necessary
export X509_USER_PROXY=/dicos_ui_home/<your account>/x509up_u<your uid>

# Setup CMSSW environment
source /cvmfs/cms.cern.ch/cmsset_default.sh

cd /.../CMSSW_X_Y_Z/src
# Note that it's not `cmsenv` but the following
eval $(scram runtime -sh)

...
```

- Jobs under Miniconda3

```
#!/bin/bash

# Setup grid certificate if necessary
export X509_USER_PROXY=/dicos_ui_home/<your account>/x509up_u<your uid>

# Setup conda environment
source /dicos_ui_home/<your account>/miniconda3/etc/profile.d/conda.sh

# Enter your emvironment space
conda activate <your env name>

...

conda deactivate
```

HTCondor configuration files need to follow the standard HTCondor syntax (More details in here). Here has a basic config file:

```
executable  = runjobs.sh
arguments   = $(infile) $(outfile)

output      = output/runjob.$(ClusterId).$(ProcId).out
error       = error/runjob.$(ClusterId).$(ProcId).err
log         = log/htc.log

request_cpus      = 1
request_memory    = 512M
request_disk      = 1G

max_retries = 1

queue infile, outfile from IORecord.dat
```

- `executable` : Your executable file, i.e. a bash or python script.
- `arguments (Option)` : Arguments that need to be inputed to your executable file.
- `output` : Create files to record output message.
- `error` : Create files to record error message.
- `log` : Create files to record HTCondor job log.
- `request_cpus (Option)` : Number of CPU cores is requested in each job. However, the cluster only allow 1 core.
- `request_memory (Option)` : Memory is requested in each job.
- `request_disk (Option)` : Storage is requested in each job.
- `max_retries (Option)` : Number of rerun when some jobs are failed.
- `queue` : Number of jobs. You can write only `queue` for a single job or `queue 10` for 10 jobs.

However, the syntax of the `queue` in the above example could be more realistic when performing parallel calculation. The HTCondor config file actually allows to include the arguments of the executable file through external files as “IORecord.dat” in the above config file. Here is the content of “IORecord.dat” as

```
input1.root output1.root
input2.root output2.root
input3.root output3.root
input4.root output4.root
input5.root output5.root
```

Namely,

```
queue infile, outfile from IORecord.dat
```

means each line in “IORecord.dat” will be assigned as `infile` and `outfile` in the config file and make a job. Therefore, total 5 jobs are submitted.

Once you finish these two files, you can submit it by

```
condor_submit example.sub
```

Your screen will display:

```
[youying@tidc-ui01 example]$ condor_submit example.sub
Submitting job(s).....................................................................................................
100 job(s) submitted to cluster 426.
```

Monitor your job status by

```
condor_q
```

Your screen will display:

```
-- Schedd: queue@tidc-ui01.grid.sinica.edu.tw : <202.140.187.218:9618?... @ 11/12/23 14:00:32
OWNER   BATCH_NAME    SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
youying ID: 427     11/12 14:00      _     75     25    100 427.0-99

Total for query: 100 jobs; 0 completed, 0 removed, 25 idle, 75 running, 0 held, 0 suspended
Total for youying: 100 jobs; 0 completed, 0 removed, 25 idle, 75 running, 0 held, 0 suspended
Total for all users: 100 jobs; 0 completed, 0 removed, 25 idle, 75 running, 0 held, 0 suspended
```

If you would like to kill submitted jobs, you can use the following commands

```
# Kill all jobs belonging to you
condor_rm <your account>

# Kill according to job id
condor_rm <job id>
```