DRAGNET Cluster Usage

Some non-obvious and DRAGNET hardware and setup specific info on using DRAGNET wrt logins, (fast) network transfers, cluster-wide commands and compute job submission / scheduling via SLURM.

Feel free to extend / improve!

Access and Login

To get an account, get permission from the Dragnet PI: Jason Hessels (`hessels@astron.nl`). With permission from Jason, ask Teun Grit (`grit@astron.nl`) to add access to DRAGNET (via NIS). If you don't have access to the LOFAR portal, tell him. He can give you access and depending on from where you want to login, add your IP to a white list.

Having an account, ssh to hostname `dragnet.control.lofar` or easier, just **dragnet**, from the LOFAR portal (or tunnel through it):

```
$ ssh USERNAME@dragnet
```

Password-less Login

Within the cluster (or even to it), don't bother typing your password all the time. Passwords make cluster-wide commands a nightmare. Instead, use an ssh key pair:

```
$ ssh-keygen -t rsa  # or copy an existing public key pair to .ssh/
$ cat .ssh/id_rsa.pub >> .ssh/authorized_keys
$ chmod 600 .ssh/authorized_keys
```

(For completeness: Your .ssh/id_rsa contains your private key. Do **not** share it with others. If compromised, asap regenerate the key pair.)

To make login between nodes more reliable, you can disable the ssh host identification verification within DRAGNET. It is overkill within our cluster and if we ever need to reinstall a node, its key fingerprint will then change, causing your (auto-)login to fail until you manually remove the offending entries from `.ssh/known_hosts`.
To disable, add to (or create) your `.ssh/config` file on DRAGNET:

```
NoHostAuthenticationForLocalhost yes

Host dragnet dragnet.control.lofar dragproc dragproc-10g
dragproc.control.lofar dragproc-10g.online.lofar drg?? drg??.control.lofar
drg??-10g drg??-10g.online.lofar drg??-ib drg??-ib.dragnet.infiniband.lofar
StrictHostKeyChecking no
```

Now test if password-less login works by logging in and out to `drg23` without entering a password (this should succeed with no output):

```
ssh drg23 exit
```

Finding Applications

To use most applications conveniently, you need to set or extend environment variables, such as PATH, LD_LIBRARY_PATH, PYTHONPATH, … Unlike CEP clusters that use the home brew `use <pkg>` command, we use the `module <command> [pkg]` command. (Some users just export the needed values explicitly.)

Practical Summary

On DRAGNET add to your .bashrc e.g.:

```
module add local-user-tools lofar casacore
```

or a similar list (`casacore` contains python-casacore aka pyrap).

Command to print the list to select from:

```
$ modules avail
```

Re-login (or enter the `module add <pkgs>` command) to apply in each login session. (If you use the screen(1) program, restart it too!)

If you want to keep using the same tool version instead of auto-upgrading along when updates are installed, then specify the versioned module name (when available), e.g. `lofar/2.17.5` or `casa/4.6`

Using the Environment Modules

The "environment values" is a set of key-value pairs per program, inherited from the program that started it. Each shell has its own copy (so if you change one, others are unaffected). Your environment is copied and adjusted at login. You can further adjust it in .bashrc (Note that there is also .bash_profile and .profile. What to change for different login types varies among Linux distros and shells and documentation is not always matching reality…)

The complete, sorted list (1000s of lines) and (unexported) shell variables can be printed by typing `set`.

Type `module help` for a list of `module` commands.

List of available modules (Sep 2016):

```
$ module avail

------------------------------------------------------------------
/usr/share/Modules/modulefiles --------------------------------------------------
```

```
-----------------------
dot           module-git  module-info modules     null         use.own


----------------------------------------------------------------------------
/etc/modulefiles -----------------------------------------------------------
-----------------
aoflagger/2.8.0    casacore/2.0.1    casarest/1.4.1    cuda/8.0
lofar/2.11.4       lofar/current     mpi/openmpi-x86_64
aoflagger/current  casacore/2.0.3    casarest/current  cuda/current
lofar/2.12.0       lofardal/2.5.0    srm/2.6.28
casa/4.6           casacore/2.1.0    cuda/7.0          karma/1.7.25
lofar/2.14.0       lofardal/current  wsclean/1.12
casa/current       casacore/current  cuda/7.5          local-user-tools
lofar/2.17.5       mpi/mpich-x86_64  wsclean/current
```

Add latest lofar module to your env:

```
$ module add lofar   # or a specific one e.g. module add lofar/2.17.5
```

Remove module from your env (e.g. if it conflicts with another version you want to use):

```
$ module rm lofar
$ module purge  # remove all added modules
```

To run the prefactor and factor imaging pipelines, you may want to only use the following command (do not add casa). (And ensure your pipeline.cfg refers to the same paths.)

```
$ module add local-user-tools wsclean/1.12 aoflagger/2.8.0 lofar/2.17.5
casarest/1.4.1 casacore/2.1.0
```

If you login and want to use CASA instead, better run /usr/local/casa-release/bin. You may also remove (i.e. purge) all added modules and add the casa module, but it only sets PATH, which then may find CASA's own bin/python and bin/ipython, which interferes easily with other tools.

See what adding the local-user-tools module does (Aug 2016):

```
$ module show local-user-tools
-------------------------------------------------------------------
/etc/modulefiles/local-user-tools:

module-whatis     Adds tools, libraries and Python modules under /usr/local
to your environment.
  Pulsar tools : dspsr, psrcat, psrdada, pstfits, psrchive, tempo, tempo2,
dedisp, sigproc, ffasearch, ephem, see, clig, ...
  Imaging tools: factor, losoto, ds9, Duchamp, sagecal, excon imager,
rmsynthesis, pyselfcal, ...
prepend-path      PATH /usr/local/bin
prepend-path      PYTHONPATH /usr/local/lib/python2.7/site-
packages:/usr/local/lib64/python2.7/site-packages
-------------------------------------------------------------------
```

Cluster-wide Commands

To run a command over many cluster nodes, use `cexec` (as on CEP2/3), `ansible`, or a shell loop around an ssh/scp command. (First, see the section above on **Password-less Login**.)

- cexec (shell) runs any shell command in parallel. Output is sorted and only appears after all nodes finished. Indexed hostname specification.
- ansible (Python) is easy with simple commands or with Ansible modules to support idempotent changes. Easy integration in Python programs. No sorted output, but node output appears when a node is done. No shell interpretation of commands, which may be a restriction or rather safe. Can run commands in parallel. Tailored for system administration, configuration and deployment.
- shell loop around ssh is most basic and possibly powerful wrt UNIX tools, but tricky wrt escaping, which remote environment values are actually used, and for dealing correctly with filename corner cases. Scripts easily end up shell specific (e.g. bash vs tcsh).

NOTE: be careful with potentially destructive operations like `rm -rf`. Accidents have happened (data loss) on CEP2 with cexec and shell scripts.

C3 Cexec

The [Cluster Command and Control](#) (C3) tool suite contains the cexec(1) program that can be used to run commands over many nodes.

Example:

```
$ cexec drg:3-5 "df -h"    # disk usage on the drg04(!), drg05, drg06(!) nodes
$ cexec dragnet:23 ls      # run ls on dragproc
$ cexec hostname           # hostnames as seen from each cluster node
```

The hostname specifier (2nd optional argument) must contain a ':' and may also be `drg`, which excludes the `dragproc` node. The `dragnet` hostname specifier contains all nodes (incl head node). The `drg` group is without `dragproc`. The head node is never part of the group, though you can explicitly specify it if needed e.g. in scripts. Note that the hostname numbers here specify start and end index (starting at 0!).

Ansible

[Ansible](#) is a tool to automate cluster (administration) tasks.

Examples of simple commands:

```
$ ansible alldragnet -a 'df -h'                          # disk usage on all nodes
$ ansible proc:workers -f 25 -a 'df -h /data1 /data2'       # disk usage on dragproc and worker nodes, connect to max 25 nodes at a time
$ ansible workers -f 25 -a 'ls -al /data1/LOBSID /data2/LOBSID'  # list
```

```
/data*/LOBSID files on all drg* nodes, connect to max 25 nodes a time
$ ansible drg01:drg17 -a 'ls -l /data1'                    # list
/data1 on drg01 and drg17 (not drg01 till drg17)
```

Apart from hostnames, the following hostname groups are also recognized on DRAGNET: `head`, `proc`, `workers`, `alldragnet`, `all` (last two are the same). The command must be a simple command. It can be the name of an executable shell script if accessible to all hosts, but not a compound shell command with &, &&, pipes or other descriptor redirection (you can of course run the shell with some argument, but then, what's the point of using ansible like that?).

Background: Ansible heavily relies on the idea to specify what you want in terms of the desired situation rather than what to do to get there. Such *idempotent* commands work correctly regardless whether some nodes are already ok or different. To this end ansible has numerous modules to manipulate system settings in an easy way, but you can also write your own modules (e.g. to reinstall (parts of) a type of node), or so-called *playbooks* to manage configuration and deployment.

For many common system admin related tasks, use an ansible module. Search the Ansible Module Index for more info.

Shell Loop and SSH

Examples:

```
$ for host in $(seq -f drg%02g 1 10); do ssh $host "hostname && df -h /data1
/data2"; done  # disk usage on the drg01-drg10 nodes
$ for host in drg01 drg17; do ssh $host "df -h"; done
# disk usage on drg01 and drg17
```

Be careful with complex commands!

Data Copying

Generic data copying info plus cluster specific subsections.

To copy large data sets between nodes or into / out of DRAGNET, you can use `scp` or `sftp` or `rsync`. However, these tools are unable to fill links well in excess of 1 Gb/s. For multiple large files, you can start several transfers, but this may not be enough and is tedious. Single core CPU power may also be a problem. To alleviate CPU load, select the simple `arcfour` cipher (I consider the LOFAR network private enough for any security risk to materialize).

```
$ scp -B -c arcfour <src_node:path> <dst_node:path>
```

The `bbcp` tool is able to utilize more bandwidth. It first uses `ssh` to login and then starts `bbcp` on both sides. Example command we use to copy all files in a directory to CEP3 node lof003:

```
$ bbcp -A -e -s 4 -B 4M -r -g -@ follow -v -y dd --
drg23-10g.online.lofar:/data1/xxx/cs/
lof003.offline.lofar:/data/projects/xxx/Lyyyyyy/cs/
```

Notes:

- OpenSSH-6.7 no longer allows the `arcfour` cipher, but DRAGNET uses 6.6. (Both sides of the transfer must allow it.)
- The `rsync` tool remains great to retransfer (or check if) data changed in minor ways, as `rsync` only syncs (chunks around) the changes. `rsync` is also great to transfer many small files.
- For `bbcp`, if you want to see network speed, drop the "-y dd" option to flush.

Hostname Hell and Routing Rampage

If you are just running some computations on DRAGNET, skip this section. But if you need fast networking, or are already deep in the slow data transfers and rapid-fire connection errors, here is some info that may save you time wrt the multiple networks and network interfaces. (Or just tell us your needs.)

Hostnames

Control network:

- dragnet(.control.lofar)
- dragproc(.control.lofar)
- drg01(.control.lofar) - drg23(.control.lofar)

10G network:

- dragproc-10g(.online.lofar)
- drg01-10g(.online.lofar) - drg23-10g(.online.lofar)

Infiniband network (~54G):

- drg01-ib(.dragnet.infiniband.lofar)

(There is also a 1 Gb IPMI network.)

Note that for copying files between hard disks, there is some benefit to use the 10G network. If you have data to copy on `/data1` and `/data2`, transfer from/to both areas in parallel. This will not reach 10 Gb/s, so using infiniband for such transfers does not help.

Cross-Cluster

When writing scripts that (also) have to work cross-cluster, prefer to use the fully-qualified domain names (FQDN) (e.g. `drg23-10g.online.lofar` instead of just `drg23`). See `/etc/hosts` on any node for the list.

In most cases, you will use the network as deduced from the destination hostname or IP. Indicate a 10G name to use the 10G network. Idem for infiniband. (Exception: CEP 2, see below.)

Note: Copying large data sets at high bandwidth to/from other clusters (in particular CEP 2) may interfere with running observations as long as CEP 2 is still in use. If you are unsure, ask us. It is ok to

use potentially oversubscribed links heavily, but please coordinate with Science Operations and Support!

CEP 2

Initiate connections for e.g. data transfers from CEP 2 to HOSTNAME-10g.online.lofar to transfer via 10G.

The reverse, connecting from DRAGNET to CEP 2, by default will connect you via DRAGNET 1G (e.g. for login). To use 10G (e.g. to copy datasets), you need to bind to the local 10G interface name or IP. The program you are using has to support this via e.g. a command-line argument.

CEP 3

Use the `lofXXX.offline.lofar` hostnames to transfer via 10G.

CEP 4

CEP 4 has a Lustre global file system. Copying data to DRAGNET is supposed to happen via `lexar003.offline.lofar` and `lexar004.offline.lofar`.

A Lustre mount has also been set up on DRAGNET, but the storage name is not mounted by default.

External Hosts (also LTA Staged Data)

To copy data sets from outside the LOFAR network (e.g. staged long-term archive data) into DRAGNET, there is unfortunately only 1 Gbit/s available that is shared with other LOFAR users. A 10G link may become available in the future.

There are 3 cases to distinguish:

1. Access external hosts (but not lofar-webdav.grid.sara.nl) from the LOFAR network

This all uses the LOFAR portal / public internet link (1 Gbit/s). Since the LOFAR portal is used by all users to login, it is important not to overload it. Load is monitored and too hungry copying processes may be killed if they harm other users.

So please rate-limit your download from outside into DRAGNET and CEPx! A reasonable chunk of 1 Gbit/s is 400 Mbit/s (= 50 MByte/s), such that if somebody else does this too, there is still a bit of bandwidth for dozens of login sessions from other users. (Yes, this is hardly a foolproof strategy.) Please use:

```
$ scp -l 400000 ...        # value in kbit/s
or
$ wget --limit-rate=50m ... # value in MByte/s
```

```
or
$ curl --limit-rate=50m ... # value in MByte/s
or
$ rsync --bwlimit=51200 ... # value in kByte/s
```

For those interested, you can use `atop 2` on the LOFAR portal as a regular user to see the currently routed traffic rate across the network interfaces. More details on a single DRAGNET node can be monitored by administrating users using the `nethogs` program. Everyone can see a lot of cluster performance metrics on http://ganglia.astron.nl/ (select `dragnet`).

2. Download via http(s) from lofar-webdav.grid.sara.nl to the LOFAR network

A http(s) `squid` proxy server has been set up to forward the traffic over a special line to SurfSara. This activates when you set the `http_proxy` and/or `https_proxy` environment variable correctly before starting the download, like this:

```
# for http URLs
http_proxy=lexar004.control.lofar:3128 wget --no-check-certificate
http://lofar-webdav.grid.sara.nl/...

# for https URLs
https_proxy=lexar004.control.lofar:3128 wget --no-check-certificate
https://lofar-webdav.grid.sara.nl/...
```

However, atm you need to authenticate to this proxy. Get an account via the ASTRON "Science Operations & Support" group sos@astron.nl (sigh...)

If you use this only for lofar-webdav.grid.sara.nl, you do not need to rate-limit your downloads as specified above.

Note: This also works for other http(s) destinations than SurfSara servers, however, then you need to rate-limit your http(s) traffic as described above under **1.**. Do **not** use this for other LTA sites than SurfSara, as atm this interferes with data streams from some int'l station!

3. Between ASTRON internal 10.xx.xx.xx nodes and the LOFAR network

Specifically for ASTRON hosts with an internal `10.xx.xx.xx` IP, you can access nodes in the LOFAR control network directly by IP to copy data at 1 Gbit/s without going through the portal.lofar.eu node. There is no need to rate-limit this, the network will divide bandwidth among transfers when needed.

SLURM Job Submission

To utilize the cluster efficiently, we use the [SLURM workload manager](). This is also supposed to ensure that batch jobs do not interfere with observations that DRAGNET participates in (as in: micromize observation data loss).

Random notes:

- SLURM does not enforce accessing nodes through it; one can access any node via ssh. Depending on the intention and the current workload, that may be fine or less desirable.
- SLURM has a ton of options that we haven't all set up. In particular, atm it does not enforce exclusive access to GPUs via cgroups (although it does set CUDA_VISIBLE_DEVICES if you explicitly request GPUs). Once a node is (partially) assigned to your program, your program can in principle use any resource on that node.

Introduction: the trivial stuff

From any DRAGNET node (typically the `dragnet` head node), you can submit compute (or perhaps also separate data transfer) jobs.

Use `srun` to start a task, see output as it is produced, and wait for completion. Use resource options such as –nodes=10 or –tasks=10, and/or –nodelist=drg01 to reserve nodes or CPUs (see below or `man srun` for more info).

```
$ srun --nodes=5 --nodelist=drg01,drg02 ls -l /data1 /data2
dir1 dir2 file1 file2 [...]
```

Use `sbatch` to queue a job to run a supplied batch script with various commands, advanced options, and resource specifications in shell comments (see below). (No need to also use the `screen` command.) Slurm immediately prints the JobId and returns. It redirects stdout and stderr to a slurm-<JobId>.log file. For simple cases, auto-generate the script using –wrap.

```
$ sbatch --mail-type=END,FAIL --mail-user=your-email-addr@example.com --
wrap="ls -l /data1 /data2"
Submitted batch job <JobId>
```

The `srun` and `sbatch` mostly take the same args, so likely, you want to combine the 2 examples above using `sbatch` and the resource options, or better, supply a simple shell script.
Tip: use absolute path names and $HOME.

Show list of jobs queued:

```
$ squeue
         JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
             9   workers       ls amesfoor CD       0:01      1 drg
```

Show list of recently completed jobs:

```
$ squeue -t COMPLETED
         JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
             9   workers       ls amesfoor CD       0:01      1 drg
```

Show details of a specific job:

```
$ scontrol show job <JobId>
JobId=223058 JobName=wrap
    [<~20 lines of info on status, resources, times, directories, ...>]
```

Show list and state of nodes. When submitting a job, you can indicate one of the partitions listed or a (not necessarily large enough) set of nodes that must be used. Please hesitate indefinitely when trying to submit insane loads to the head partition. :)

```
$ sinfo --all
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
workers*    up    infinite    24   idle dragproc,drg[01-23]
head        up    infinite     1   idle dragnet
lofarobs    up    infinite    24   idle dragproc,drg[01-23]   # Note: for
observations; use 'sinfo --all', else only usable & visible for users in
lofarsys group (+ slurm,root)
```

If you get an error on job submission that there are no resources in the cluster to ever satisfy your job, and you know this is wrong (no typo), you can see with the sinfo if there are nodes out of service. (SLURM may remove a node from a partition on misconfiguration or hardware malfunctioning.)

More detail:

```
$ sinfo -o "%10N %8z %8m %40f %10G %C"
NODELIST   S:C:T    MEMORY   AVAIL_FEATURES                           GRES
CPUS(A/I/O/T)
dragnet,dr 1+:4+:1  31800+   (null)                                  (null)
0/20/0/20
drg[01-23] 2:8:1    128500   (null)                                  gpu:4
0/368/0/368
```

where in the last column A = Allocated, I = Idle, O = Other, T = Total

Hints on using more SLURM capabilities

The sbatch(1) command offers to:

- take a user-supplied job (batch) script, not just to start your script, but also to set up a job array or workflow
- have stdout/stderr go to a file
- copy the program (and possibly library and data dependencies) to the to be used nodes
- run the job without blocking your terminal on its completion. This is useful for e.g. substantial processing jobs
- auto-restart on failure (not sure when/how that applies)

Apart from nodes, it is also possible to indicate scheduling constraints on CPU cores, GPUs, memory, or network bandwidth (if we set that up).
Atm, you have to indicate constraints for:

- either number of nodes or CPUs
- number of GPUs, if any needed. If no GPUs are requested, any GPU program will fail. (Btw, this policy is not fully as intended, so if technically it can be improved, we can look into it.)
- if you want to run >1 job on a node at the same time, memory. Just reserve per job: 128500 / NJOBS_PER_NODE. By default, SLURM reserves all the memory of a node, preventing other jobs from running on the same node(s). This may or may not be the intention. (If the intention, better use --exclusive.)

Note that a CPU is to SLURM a hardware resource that the OS can schedule a task on. On DRAGNET it is a CPU core (16 on all nodes, but 4 on the head node). (On typical SLURM installs, it's a hardware thread, but we don't expect to get something out of HyperThreading.)

To indicate a scheduling resource constraint on 2 GPUs, use the –gres option (*gres* stands for *generic resource*):

```
$ srun --gres=gpu:2 -n 1 your_gpu_prog
```

To indicate a list of nodes that must be used (list may be smaller than number of nodes requested). Some examples:

```
$ srun --nodelist=drg23 ls
$ srun --nodelist=drg05-drg07,drg23 -n 8 ls
$ srun --nodelist=./nodelist.txt ls   # with a '/' in the arg value
```

For the moment, see more explanation and examples at
http://hpcf.umbc.edu/how-to-run-programs-on-maya/

Please see the manual pages on srun(1), sbatch(1), salloc(1) and the SLURM website for more info.

SLURM Cluster Management

Some commands I looked up and probably need again another time.

Bring fixed node back to partition from state DOWN to state IDLE (logged in as slurm):

```
$ scontrol update NodeName=drg23 state=idle
```

Users can resume their (list of) job(s) after SLURM found it/they cannot be run (network errors or so) and sets the status to something like 'launch failed, requeued held'. If the range is sparse, slurm prints some errors, but does resume all existing jobs.
This can also be exectured by users for their own jobs.

```
$ scontrol resume 100
$ scontrol resume [1000,2000]
```

From:
https://www.astron.nl/lofarwiki/ - **LOFAR *Wiki***

Permanent link:
https://www.astron.nl/lofarwiki/doku.php?id=dragnet:cluster_usage&rev=1490200723

Last update: **2017-03-22 16:38**