REALTA Users Guide

Docker Manual

This chapter will give you a rough overview of how to use the Docker containers installed on the UCC node.

Getting Started with Docker on the REALTA Nodes

Getting Started with Docker on the REALTA Nodes

As of right now, all of the REALTA compute nodes (UCC*) and storage node (NUIG1) have some form of the standard I-LOFAR docker image. The NUIG1 machine has a slightly outdated version without GPU support (due to the lack of a CUDA device in that machine), but they should all contain the software you need to perform pulsar (or other realtime sampled data) processing.

The source of the image can be found here, and describes what software and which versions have been installed to the image. While you are free to install or change software in the image at runtime (you have root access), if you have a request for an update to existing software, or something new you think should be in the global image, let David McKenna know.

One Line Quickstart

If you just want to jump in and get started, here's a command you can use. It's advised to bind this to an alias to make it easier to call. It is current on the obs account under "dckrgpu".

```
# UCC Nodes
docker run --gpus all -e TERM -v /mnt:/mnt --rm -it pulsar-gpu-dsp2020


#NUIG Node
docker run -e TERM -v /mnt:/mnt --rm -it pulsar-dsp2020
```

These commands will launch a container container that will clean itself up when you exit a session.

As a result, it's advisible to run it within a tmux shell (`tmux new -S docker_workspace`) or screen (`screen -S docker_workspace`) to maintain it between work sessions. To deattached from these, you can use `Control+B,D` for tmux and `Control_A,D` for screen.

To exit a docker session, just use the standard bash `exit` command in the docker shell window.

A Note on File Permissions

Any files created or modified within the docker container will become owned by the vritual root user. As a result, you may want to reclaim these files to your REALTA user account after you're finished in the container.

To do this, you will need to get your user ID and group ID on each node via the `id -u` and `id -g` commands (or `ll / ls -l` in a directory you own and look at the IDs). Afterward, you can reclaim the file via the `chown` command, running within any docker container:

```
root@b68e2344da1>:~/$ ls
my_working_file my_working_dir
root@b68e2344da1>:~/$ chown <realuid>:<realgid> ./my_working_file
root@b68e2344da1>:~/$ chown -R <realuid>:<realgid> ./my_working_dir/
```

Other Flags

There are a few flags you might want to get familiar with and add in to the command above.

More `-v /ref/to/host:/mount/in/container` mounts can be of use, such as adding access to your home directory through the docker with `-v /home:/home_local`

`--cpuset-cpus="<procids>"` and `--cpuset-mems="<numa nodes>"` can be useful to ensure that your code is run in a single NUMA domain, given that `numactl` is not available within containers

Docker Software

While the containers allow you to install any software you want without effecting other users on the system, a large amount of software is already provided within the containers, focused around working with raw REALTA data and processing pulsar or transient target observations. This chapter will act as a rough outline of the software available in the container and their use.

Unless otherwise noted, the software has been frozen in the state it was on February 22nd, 2020 to prevent problems between dependancies during the image compilation.

Generic Software

Pulsar Software

Transient Software

Pulsars

Transients and Single Pulse Detection

Observing Software

NUMA-Aware Update

Given we're often stuck with a full drive or want to start transferring data to collaborators while observations are on-going, I had a look into the NUMA setup of UCC1 to see if we could try keep recording processing on one NUMA node and offloading on another node.

Jargon

NUMA: Non-uniform memory access -- consider that we have dual CPU machines, these are split into 2 NUMA nodes to reflect the increase in latency between accessing something in the cache or memory attached to one of the CPUs from another.

The System Overview

top.png

Each of the REALRA nodes has two NUMA nodes, each with different components attached to them via the PCIe bus. We can see that node #0 has

- The fibre connections (eno* devices)
- The Tesla V100 (card0)

While node #1 has

- The storage devices (sd*)
- The infiniband networking card (ib*)

This is almost ideal for what we want to achieve, as we can restrict the recoridng processes to node #0 (with the fibre card having DMA to node #0 as a result, but writing to the drives at node #1) while other processes can execute access the disks on node #1.

As a result, I modified the normal `generic_ucc1.sh` script to constrain Olaf's recorder to NUMA node 0 using `numactl`,

```
numactl -m 0 /home/obs/joe/Record_B1508/$recording_program --ports 16130 ... &
numactl -m 0 /home/obs/joe/Record_B1508/$recording_program --ports 16131 ... &
```

```
numactl -m 0 /home/obs/joe/Record_B1508/$recording_program --ports 16132 ... &
numactl -m 0 /home/obs/joe/Record_B1508/$recording_program --ports 16133 ... *
```

This mean the recorder will allocate memory and only perform processing on NUMA node 0. This include any child processes used by zstd for compression, though we could modify the soruce to force that process to stay on NUMA node #1 if we want it to have direct, priority access to the disks.

Some other kernel parameters were also changed for this test, namely the size of the UDP buffer queue and the maximum size. These were increased from 26kb (~1 packet/port) and 1,000 respectively to ~250MB (~0.1 seconds of data/port) and ~500k packets, though the increase in queue length was likely not needed as the kernel never reported a value higher than 200.

With these changes, two obsevrations were performed while an rsync command was moving data from UCC1to UCC3 across the infiniband network, while being forced onto node #1. Packet loss was minimal during the first observation, with the worst port losing 2,105 packets across a 7 hour observation (0.000650% packet loss). The transfer finished a few hours in to the second observation, with the worst port losing 1,726 packets across the 6.5 hours (0.000542% packet loss).

While these values are low, by plotting the packet loss during the second observation it is extremely clear when the transfer stopped occurring:

packetloss.png

However, with the reduced packet loss levels (2,000 packets corresponds to 0.16 seconds) this setup should allow us to perfrm some amount of data transfer off UCC1. However I am unsure as to the effect of remote transfers as these will likely traverse the fibre line which will likely elevate the packet loss even further, though I'll say it's worth looking in to the loss rates during one of the upcoming local modes.

Processing Methodologies

The standardised methods for processing I-LOFAR data produts

Single Pulse Source Observations

FRBs and RRats are highly transient by their nature. As a result, we could see a single peak in a 10 hour observation. While keeping the raw voltages on hand can help with re-processing observations to avoid missing features due to issues in the current methodology, we no longer have the storage space on the UCC processing nodes or NUIG1.

Proposed methodology for processing single pulse observations observations:

- Process the observation with **CDMT**. Produce both a **0-DM and N-DM** filterbank at the nominal time resolution (currently 655.36us, 16**x ts, 8x chan**)
- Perform **RFI detection** on the 0DM filterbank
  - We currently do not have a strong methodology here, apart from bandpass analysis and rfifind for detecting DM=0 features
- Search this output after an **8-bit decimation from digifil**
  - Log the heimdall commands used, RFI channels flagged
  - Investigate the optimal scale timescale for digifil (-I, default 10s) for FRBs; some are expected to last up-to or over 1 second at our frequencies due to scattering.
- After a search is complete, **archive the CDMT filterbanks**
  - **Digifil: 2x ts** for further space savings if needed, mostly a layover from previous 8x tsch
    - -I 0 : No scale changes, raw 2:1 conversion
    - -b-32 : Float32 output, no change from raw filterbank
    - -t 2 : Down sample to 655us resolution
  - **Compressed** with zstandard: Further 10-20% compound storage saved

There are a few ways that this methodology could be changed to make the resulting filterbanks easier to search + store, or improve SNR

- Future changes
  - **Chop bandwidth**? Top 5MHz / Bottom 7 are Nyquist suppressed + RFI contaminated
    - Removing these could save us 15% of storage and speed up processing as searching the last 10MHz introduces an addition delay of 25 seconds @ R3's DM
    - No easy way to do this with the current voltage extraction/processing method, would need to be after the filterbanks are formed
  - Investigate having CDMT **split filterbanks** every N samples

- Consider overlap requirements to not miss signals on the boundaries
- Duplicated data, but higher theoretical SNR when we can include more channels by more selective RFI flagging
- Or just find a decent RFI flagging algo...

We note that for RRats, we do not recommend forming a 0DM filterbank as those sources often do not need validation as they should be bright enough to be obvious with/without coherent dedispersion.

Step	Method	Storage Used	Product	Overall on Disk
Generate Voltages	Observer	1	1	
Compressed	zstandard, Olaf's recorder	~0.6-0.8	0.6	0.6
CDMT	-a -b 16 -d 0,DM,2	0.125	0.125	0.725
Digifil (Search)	-b 8 -I <DECIDE>	0.03125	0.03125	0.75625
Cleanup: Digifil (search)	rm	-0.03125	-0.03125	0.725
Compress CDMT (compress)	zstandard	~0.1	0.1	0.825
Cleanup: Voltages, CDMT	rm	-0.6 - 0.125	-0.725	0.1
Overall				**~100 GB/obs-hr**

LuMP Processing

An alternative recording method for some observations is with the LuMP software from MPIfRA. It has been used in the past for coordinated observations with FR606 and at the request of observers from the UK and Poland.

Any DSPSR sub-program (dspsr, digifil, digifits) can be used to process a LuMP observation, but each port/process (if using a multi-processed recording mode) must be processed separately and then combined (fils: filmerge, fits/ar: psradd).

So as an example, to process with digifil you may choose to process a set of observations using the command

```
for file in *.raw; do
digifil -b-32 -F <NUMCHAN>:D -I 0 -c -set site="I-LOFAR" $file -o $file".fil"
done


filmerge *.raw.fil
```

To perform coherent dedispersion (`-F <CHAN>:D`) for a known pulsar target (inside LuMP metadata), without any bandpass/temporal offsets (`-I 0 -c`), producing a 32-bit output (`-b-32`) filterbank.

Many issues arise with modern versions of DSPSR when processing raw data, not limited to the dedispersion kernel failing, the default filterbank failing, misaligned folds when directly processing with DSPSR, etc. As a result we use a modified version of the workflow presented above for processing a typical LuMP observation.

```
baseName=$1


# Process the raw data with digifil. Perform 8x channelisation, 2x time scrunching (tsamp ~ 81us)
# Fake machine to COBALT as sigproc's filmerge will refuse to merge fils if the header is FAKE
for fil in *.00.raw; do digifil -b-32 -I 0 -c $fil -set machine=COBALT -set site=I-LOFAR -t 2 -F 328:1 -o $fil".fil" &
echo "hi" ; done; wait;
for fil in *.01.raw; do digifil -b-32 -I 0 -c $fil -set machine=COBALT -set site=I-LOFAR -t 2 -F 328:1 -o $fil".fil" &
echo "hi" ; done; wait;
for fil in *.02.raw; do digifil -b-32 -I 0 -c $fil -set machine=COBALT -set site=I-LOFAR -t 2 -F 320:1 -o $fil".fil" &
echo "hi"; done; wait;
```

```
# Each port should have the same number of samples and starting MJD; merge each of them
filmerge ./udp16130*raw.fil -o "./udp16130_"$baseName".fil"
filmerge ./udp16131*raw.fil -o "./udp16131_"$baseName".fil"
filmerge ./udp16132*raw.fil -o "./udp16132_"$baseName".fil"
filmerge ./udp16133*raw.fil -o "./udp16133_"$baseName".fil"

for fil in udp*"$baseName".fil; do digifil -b 8 $fil -o $fil"_8bit.fil"; done

# Fold the data, 1024 bins, ~3 secnd integration (change turns as needed)
for fil in *_8bit.fil; do dspsr -turns 4 -nsub 512 -t 4 -b 1024 -skz -skzn 4 -k IelfrHBA -O $fil"_fold" $fil; done

# Attempt to combine the data. This will not work 90% of the time due to packet loss, but worth trying.
psradd -R *.ar -f $baseName".ar"
```

PRESTO Timing

PRESTO can be used for generating timing files for use with tempo(2).

To start, a standard `prepfold` command should be run, though to use the output archives for timing the `-nosearch` flag must be used, as a result you will need a well-timed target (good entry in psrcat) or an existing ephemeris file on hand for the folding.

Once you have a `-nosearch` pfd generated, you can use the `get_TOAs.py` script to generate TOA .tims to process with tempo(2).

Timing With Tempo2 (Empty)

Processing Non-Pulse-Based Observations

The backend used for CDMT is also available in a CLI, `lofar_udp_extractor`, which is installed on the Docker containers available on the REALTA nodes.

This guide assumes you have a UDP recording (compressed or uncompressed) from Olaf Wucknitz's VLBI recording program (standard for observing with I-LOFAR) and will explain the standard operating modes, and workarounds for issues with the `lofar_udp_extractor` program. The full, up to date documentation for the CLI can be found here.

Standard Usage

```
lofar_udp_extractor \
    -i /path/to/raw/udp_1613%d.TIMESTAMP.zst \
    -o /output/file/location \
    -p <procMode>
```

This sets up the program to take a compressed ZST file, starting at port 16130 and iterating up to port 16133, outputting to the provided location in a set processing mode. Some processing modes have multiple outputs, and will require '%d' to be in the output name as a result. The most useful processing modes are

Mode ID	Output (Stokes)	Tsamp (us)	Outputs
100	I	5.12	1
104	I	81.92	1
150	I, Q, U, V	5.12	4
154	I, Q, U, V	81.92	4
160	I, V	5.12	2

164	I, V	81.92	2

Modes 150+ are only available in more recent versions, and may error out of the docker containers have not been updated recently.

There are several other useful flags for processing data, such as `-u <num>` which will change the number of ports of data processed in a given run, `-t YYYY-MM-DDTHH:MM:SS -s <num>` or `-e <file>` can be used to extract a specific chunk of time, or specify a file with several time stamps and extraction duration (with the requirement that these regions do not overlap).

The `-a "flags"` flag passes flags to mockHeader which generates a sigproc-compatible header of metadata about the observation. This can make handling Stokes data easier later on, through the use of sigpyproc for loading and manipulating data, though as of right now it is not possible to set a per-subband frequency as is needed for mode357, so a dummy fch1 (central top frequency) and foff (frequency offset between channels) should be used instead.

As an example, during a processing run on 29/10/20 of some Solar Mode357 data, the following command was used.

```
lofar_udp_extractor \
    -i /mnt/ucc1_recording/data/sun/20201028_sun357/20201028090300Sun357/udp_1613%d.ucc1.2020-10-28T09\:05\:00.000.zst \
    -o ./2020-10-28T09\:05\:00_mode357_StokesVector%d.fil \
    -p 164 \
    -a "-fch1 200 -fo -0.1953125 -tel 1916 -source sun_357"
```

Known Issues and Workarounds

When recording starts later than the supplied start time, Olaf's recorder may pick up stale packets in the UDP cache and record them at the start of your observation. This will manifest itself as a **segfault when trying to process the start of an observation**, as the program will run into issues attempting to align the first packet on each port. As a workaround, use the `-t YYYY-MM-DDTHH:MM:SS` flag to set a start time shortly after the actual data begins recording, at which point the software will be able to accurately align the packets as needed,

Getting TOA Measurements from Single Pulses

This page describes the process to get a TOA measurement for a single pulse, assuming

- You know the rough TOA of the pulse
- The input data is a Sigproc Filterbank
- DSPSR and PSRCHIVE (with GUI) are available

Many steps of this process are automated on REALTA using this python script[gist].

<getting the .ar>

Generating a Noise-Free Model

We will use the `paas` tool to generate a noise free model, which will then be used for cross-corrlation or other analysis methods to determine the pulse TOA. Choose your brightest or most characteristic pulse and being the fitting process by running

```
paas -i \ # Interactive fitting
-d /xwin # Visual GUI of choice
    <input .ar> # Input profile to use as a reference
```

Once loaded in, focus on the pulse itself by pressing `z` to set the left limit of a zoom, and left click to select the right limit. Then, left click on the left and right edges of the pulse to set the phase limits of the pulse, you will then be able to select the peak of the pulse vertically.

Once you have a rough model in the view, you can press `f` to iteratively update the model to the data, continue to update the model until you believe a good fit of the amplitude and position of the pulse has been achieved and the residuals of the region (red lines) are similar to the noise floor.

You can then quit by pressing `q`, this will save the model to disk as 3 files, `paas.m` (the model we generated), `paas.std` (an archive profile containing the shape of the model) and `paas.txt` (an ASCII copy of the model)

We will be using the `paas.std` file for determining the pulse TOAs.

Determining Pulse TOAs using the Noise-Free Model

Now that we have our archives and model, we can use `pat` to determine the pulse TOAs. We typically perform this using the following command,

```
pat -f tempo2 \ # Output in the tempo2 format
-A PIS \ # Generate cross correlations using the Parabolic interpoaltion method, chosen for the it's performance on a test dataset from J2215+45
    -F \ # Sum across frequencies before determining TOA
    -m <paas model>.m \ # Model generated by paas in the previous section
    -s <paas profile>.std \ # Archive generated by paas in the previous section
    <input archives>.ar > <output filename>.tim


# Optional flags, you may need to remove -m for these
-t \ # Plot the profile, template and residuals
    -K /xwin \ # Using an xwindow
```

The output timing file can the be used for analysis in tempo2.

SETI

Software

Charlie Giese collected his script in a git repository here.

```
git clone https://github.com/Charlie-Giese/BL_scripts
```

The rawspec channelisation software can be found here.

```
git clone https://github.com/UCBerkeleySETI/rawspec
cd rawspec
make
sudo make install
```

The turbo_seti repo can be found here.

```
pip install -U git+https://github.com/UCBerkeleySETI/blimpy
pip install -U git+https://github.com/UCBerkeleySETI/turbo_seti
```