Adapting Frequency-Hough Analysis workflow to run on IGWN resources

by S. Dal Pra, L. Silvestri,
on behalf of the
VIRGO Rome group.

HTCondor WS, 25/09/2024

Good laws already exist; it only remains to apply them.

Blaise Pascal

Every model should be as general as possible, but not more general.

Anonymous

The Frequency–Hough analysis

All-sky search for Continuous GW signals, developed by the Virgo Rome Group.

The search is on data collected during an Observing Run (8+ months) from both Livingston and Hanford detectors, for signals in the range $20\text{Hz} \leqslant F \leqslant 2048\text{Hz}$.

Input in Band Sampled Data (BSD) format (Δt=1 Month, ΔF=10Hz, $\sim 0.4\text{GB}$)

All-sky: $-90° \leqslant \beta \leqslant 90°$, $0° \leqslant \lambda \leqslant 360°$

Computing performed by `FH_analysis` Matlab executable. Very CPU intensive, several days per job, hardly predictable runtime (a few jobs take much longer than others) increasing with F

Arguments: 8 parameters

– Frequency interval $(F, \Delta F)$ with $\Delta F = 1$Hz, $F \in [20, 2048]\,\text{Hz}$

– $\text{sd}_{\min}, \text{sd}_{\max} \equiv [10^{-8}, 2 \cdot 10^{-9}]\,\text{Hz}/s$, we split this into 4 sub–intervals

– $\beta_{\min}, \beta_{\max}$ can be chosen at convenience

– $\lambda_{\min}, \lambda_{\max}$ can be chosen at convenience (recently)

O3 campaign

Performed at CNAF, strictly tied with local infrastructure assets.

In view of **O4**, effort started to run on IGWN (OSG + WLCG) Grid sites.

Problems to consider

	CNAF	**IGWN**
Job submission	HTC-CE / local	dedicated SN **AP**
BSD access	local shared fs (GPFS)	CVMFS+scitoken
Matlab RTL	local shared fs	Singularity
Max. Runtime	3 / 12 days	$\mathcal{U}[1h, 72h]$
Output upload	Grid/local w access	HTC tf data / Grid copy
Uniform APM	`gems_ap.py cnaf.json`	`gems_ap.py igwn.json`

Job Submission

- An IGWN Submit Node (Jul 2023, HTC 10.9) was set up to join the IGWN HTC pool
- A python job wrapper was written to prepare the environment for the matlab executable and generalize CNAF vs IGWN

BSD access from CVMFS

- BSD files need to be copied locally before launching the executable (it crashes on direct access from cvmfs)
- copy can fail initially, the wrapper retries a few times before give up.
- Sites can have systematic cvmfs failure; these are blacklisted in the submit file.

```
Requirements = !StringListMember(TARGET.GLIDEIN_Site,"Site1,Site2, . . . ")
```

- other sites have occasional cvmfs failure → "Site BlackHole syndrome". When this happens number of running jobs dramatically drops.

```
~]$ condor_history -lim 1000 -cons 'exitcode==2' -af \
MATCH_Glidein_Site | sort | uniq -c
    13 GATech
   922 IN2P3
     2 SURFsara
     4 PSU-LIGO
     5 USdC
    52 Vanderbilt
# Note: exitcode == 2 means "cvmfs error"
```

eday	site	okjob	kojob	ok_h	ko_h	ok_skp	ko_skp
2024-09-18	IN2P3	536	16	2642.59	2.15	737065	22467
2024-09-18	IN2P3	288	0	1501.74	0.00	423810	0
2024-09-18	IN2P3	159	7	729.97	10.99	217598	4846
2024-09-18	IN2P3	655	606	2566.31	342.26	926548	874420
2024-09-18	IN2P3	1291	736	4654.77	565.55	1754785	1096374
2024-09-18	IN2P3	335	103	1378.67	29.51	487799	142104

Matlab Runtime Library

- Matlab Executable needs to see it as a "local" folder. CNAF provides it from a shared fs, at a non standard path. Every IGWN site might or might not have its own.
- It must match the exact matlab version of the executable.
- For IGWN, that is provided via singularity (latest available: `R2023a`)

```
+SingularityImage = "/cvmfs/singular.../osgvo-matlab-runtime:R2022b"
```

Note: assuming a specific igwn image is created, that will not be providing also the MRTL. This would would need a "merge" of the two.

Output Upload

`FH_analysis` produces a Candidates datafile: `Cand_....mat`, $O(1\text{MB})$ and a few summary files.

- datafiles are transferred at CNAF storage via `gfal-copy` + `X509` proxy
- Scitoken or IAM token *could* be used BUT: have no fresh token (yet) at upload time.
- `condor_transfer_data` also available, but less uniform (i.e. different AP, different transfer_dir)

Output filename has the format: `<JobID>_Cand_<LL|LH>_<Args>_<adler32>.mat` The job with the matching `Args` attribute has actually done.

Job wrapper A python script takes care of preparing everything needed by the executable to run:

- detects Checkpoint file (more on that later)
- retrieves datafiles from the specified repository
- logs information of interest (e.g. timings)
- set custom exit code on errors, offers troubleshooting aid
- upload datafile to specified destination, clean the workspace on `exit 0`

It reads configuration from a `json` file.

With this machinery in place, jobs can be submitted to IGWN or CNAF using the same basic setup, and run successfully BUT...

The eviction problem

Initially, a vast majority, $O(90\%)$ of IGWN jobs were failing with (to me) unclear reason

- job logfile only, with a laconic "`your job was evicted`" msg
- no `stdout` nor `stderr` (no idea on how it was doing on at eviction time)

- no EvictionReason Classad Attribute.

Most reasonable reason–guess to date:

- our singlecore job is a payload started into a GlideIn pilot, which looks just like a regular 8-core job at the site where it runs.
- The GlideIn pilot usually has a maximum lifetime < 3 days (default on HTC-CEs)
- Our payload might be started into an "old" pilot having a few hours lifetime
- our pilot was tuned to last less than 3 days
- our pilot required `NumJobStarts == 0`, thus is not rescheduled after eviction

This led us to rethink out model for IGWN

Addressing Jobs Runtime limits

Jobs parameters were tuned initially to have an average runtime of 2 days. The estimation was too poor, due to beta only splitting, so we also added lambda splitting.

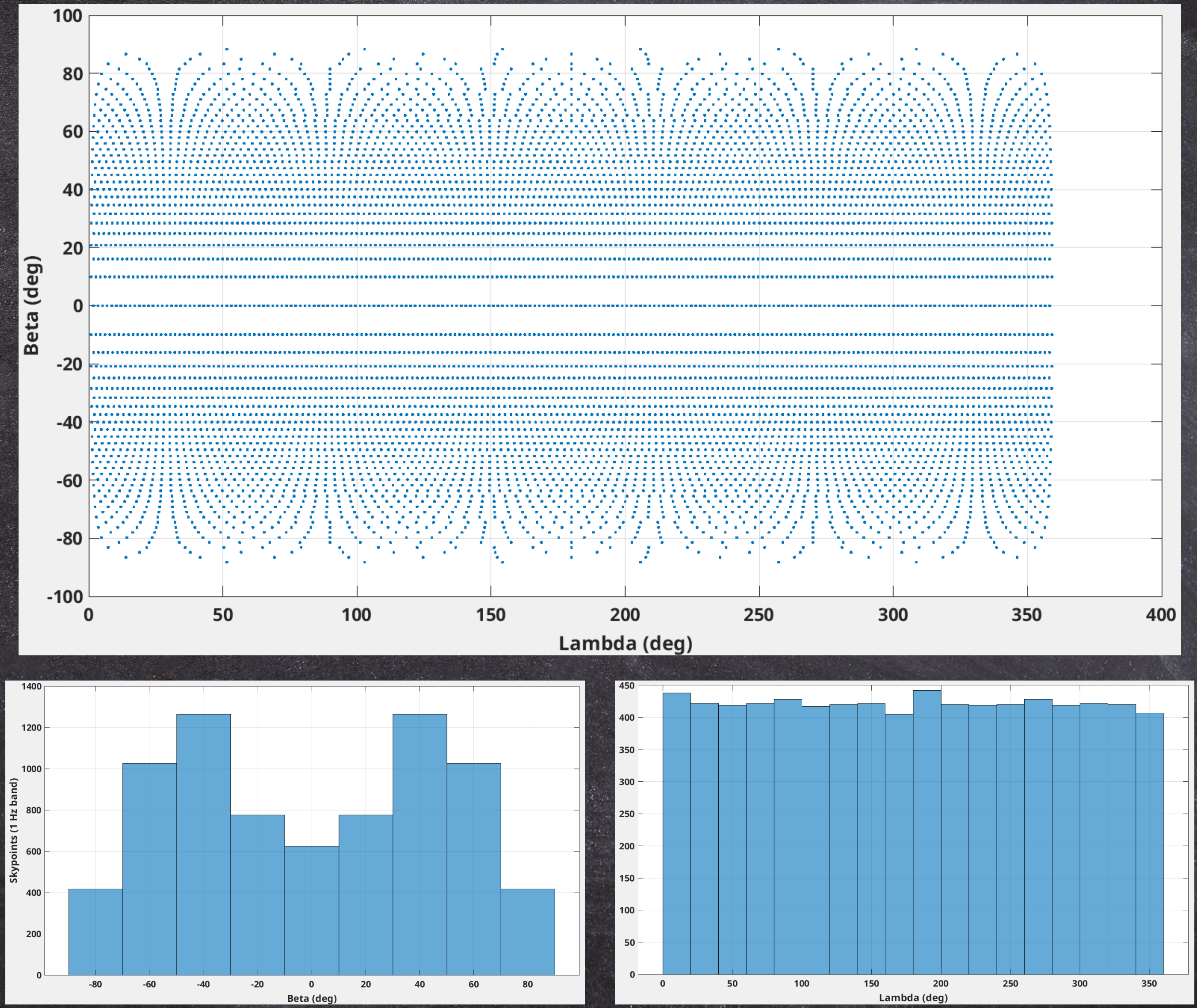

Figure 1. Skypoints for a given frequency band $F, \Delta F = 1\text{Hz}$. Points are evenly distributed in λ (longitude) but not in β (latitude)

A simple skypoints estimator was written. This allowed us to define job `Args` having ~ 2000 skypoints, which takes $\sim 40h$ on the slowest machines. This is good BUT...

Dealing with job eviction

- A HTCondor user can specify requirements for a number of cores, an amount of memory, of disk,..., Cannot require an amount of runtime.
- Well, not really:

  ```
  Requirements = (time() < (GLIDEIN_ToDie - 3600 * 48)
  ```

  (BTW: what is better, `time()` or `ServerTime`)?
- It is preferrable to have a long running job to eventually finish somehow, so we try to implement...

Checkpointing for the executable

- `FH_analysis` computes a matlab array whose length is the number of sky-points
- Array elements are computed one by one in a for loop

– We modify `FH_analysis` to dump the partial array every 50 computed sky-points to a `CKP_<args>.mat` (that's ∼ 1 dump per hour).

– at start, FH_analysis detects existence of a `CKP_<args>.mat` file. If present, that is loaded and computation goes on from there

Checkpointing

We modify the wrapper gems_ap.py to be checkpointing aware:

– If a checkpoint file exists AND `time()-GLIDEIN_ToDie < 300` it exits with `checkpoint_exit_status` (85).

– Whenever possible, HTCondor restarts the very same sandbox of the job in the same machine

That works BUT...

– Quite often the payload is restarted in the very same GlideIn pilot it just left (apparently, `STARTD` expression is not re–evaluated in this case)

– The wrapper notices that time()-GLIDEIN_ToDie < 300 is True and `exit(85)` again

– a "ping pong match" (a.k.a. race condition) begins. Three possible outcomes:

a) The GlideIn pilot is terminated while the payload is running in it and before it can `exit(85)` → the work done by the job is lost

b) The payload is restarted in a different slot in the same machine → checkpoint is succesfull, the job continues regularly

c) The payload is restarted in another machine → checkpoint partially succesfull, the job *could* continue regularly BUT...

– In case of c) the BSD files must be copied again from `cvmfs`, access is protected by scitoken BUT... The scitoken initially used has expired

– Note: the job also has a valid `X509` proxy which used to work to access `cvmfs` BUT... `X509` proxy access was disabled months ago

– The successfull case b) is pretty frequent anyway.

– This, together with reasonably short job runtime enhances success probability

After reducing average runtime and adding checkpointing, the success rate of IGWN jobs boosted.

Running a campaign

We can now organize execution of `FH_analysis` over O4a data. We have:

- `gems_ap.py` + `conf_[CNAF|IGWN|other].json` can drive execution of one job at a given pool
- `queue bulkarg from $(MY_BULKFILE)` in the submit file to specify several jobs

define the complete job list

We need to define the exact list of all the jobs to be executed. With the constraint that a job should not work on more than 2000 skypoints, it makes $3.2 \cdot 10^6$ jobs per detector. We put that list in a `PostgreSQL` database table

acct=> select id,args,statusL,statusH from o4ajobs where split_part(args,' ',1)::int = 437 limit 3;

id	args	statusL	statusH
25226	437 1 -4e-09 -1e-09 -70 -50 90 180	0	100
22686	437 1 -1e-08 -7e-09 -50 -30 180 270	100	10
22711	437 1 -7e-09 -4e-09 30 50 0 90	100	100

Select jobs to submit We select rows where `statusL` is 0 and change to 10:

```
UPDATE o4ajobs SET numtry = numtry + 1, statusL = 10
    WHERE
        id IN (SELECT id FROM o4ajobs WHERE statusl = 0
    ORDER BY split_part(args,' ',1)::int limit 100) RETURNING args;
```

The args returned by the above query are written into $(MY_BULKFILE). Concurrent agents can fetch rows without interferring to each other.

Check and submit

A cron script checks the pool with `condor_q -totals`. If less than X jobs are pending, a new bulkfile is created and queued with `condor_submit`.

Check for completed jobs

When a job output file is found at the CNAF storage, the corresponding entry is updated with `statusX=100`, and that job is completed. We scan regularly:

```
~]$ ./update_donejobs3.py
updates CNAF: {'LH': 10332, 'LL': 10398} totfiles: 295873
updates IGWN: {'LH': 23482, 'LL': 24190} totfiles: 381428
```

Resubmitting jobs After some reasonable time, jobs with `statusX=10` in the database can be set again to 0 and these will be automatically resubmitted.

Next steps

Jobs completed for both detectors are easily found from the database:

```
SELECT args FROM o4ajobs WHERE statusL=100 AND statusH=100;
```

Their ouput (two Candidate files) is compared to create one Coincidence file, for subsequent analysis.

Accounting

In the AP (`igwn-sn.cr.cnaf.infn.it`) `PER_JOB_HISTORY_DIR` has been defined to collect `history.<ClusterId>.<ProcId>` files. These are parsed to extract accounting data, which are stored in a `job` table:

month	n	fail	skp	skp_ok	wctok_h	perc_wct_fail	cnaf_wctok
2024-07	125	78	583171	228259	702.11	59.82	1193944.93
2024-08	237162	22310	282491962	249181368	575194.69	7.00	1601992.67
2024-09	419997	79594	589532449	473254601	995513.90	7.20	1785209.74

- last column is taken from CNAF accounting database
- wct_cnaf > wct_igwn however skp_day_cnaf < skp_day_igwn. This is due to a better average computing power at IGWN side.
- IGWN sites are providing more work than local submission @CNAF alone.

Conclusions

- A *slightly* general setup to run a computing campaign for CGW search has been implemented
- The `FH_analysis` campaign on O4a data is in progress, unattended, just check for problems.
- Failures with data access can occur but are not much harmful, since these happen initially, thus wasting a negligible runtime amount. Need to blacklist systematic failures.
- Failures at checkpointing are a more important loss, but still quite limited. One case can be addressed at user side.
- load distributed to different sites / Grids, more can be added at convenience.
- Still on HTC 10.9 for the AP. Moving to HTC23 would enable better solutions
- Have had very valuable help and support from the IGWN community and HTC experts, $\longrightarrow$ **THANKS!**