

A new structure for HIRLAM runs

Gerard Cats, July 2004

Introduction

In the pre-6.3.4 HIRLAM design, there were essentially two default configurations: one was for local implementations, where all HIRLAM, including mini-SMS, are run on a single host. The other default configuration was for runs at ECMWF, where all jobs were run on the workstation (*ecgate*), except the three parallel jobs (analysis, postprocessing of analysis, and forecast), and a number of jobs to copy files to or from the workstation. The latter jobs were run on the High Performance Computer (*hpca*). The distinction between the two default configurations, and the distribution over hosts, was defined in the suite definition template file (*hirlam.tdf*).

In the new, 6.3.4+, structure, all jobs will be submitted through a "Universal Job Submission Filter" (JSF). In this filter, for each job there are instructions how to submit it. So this filter now contains the information on *e.g.* on which host to run. At the same time, the system at ECMWF was changed, to run all meteorological jobs on *hpca*. The combination of these two changes greatly simplifies *hirlam.tdf*, but, of course, at the expense of having a new (Perl) script, with the code of the JSF. This script must now probably be modified for local installations.

Executing everything on *hpca* avoids the need to keep large data files on *ecgate*. I hope this will resolve the many disk problems that serious users of the HIRLAM system at ECMWF experienced. It also (almost) removes the need to use NFS-mounts, which, according to ECMWF staff, should increase stability of the system. However, currently job stdout/stderr file is still written to an NFS mount on *ecgate*; this is thought not to have a marked negative impact on stability because this file is usually small. By using NFS for this file, it is possible to inspect the file already during the execution of the job.

Some administrative jobs are still run on *ecgate*. In particular, mini-SMS runs on *ecgate*.

A few other technical changes are implemented at the same time: Archiving is now done for each file separately, without *tar*-ring the files. The files are archived in subdirectories *yyyy/mm/dd/hh* of the main experiment archive (*HL_EXP*). The script to access this archive (*Access_lpfes*) has been extended with options to make and (recursively) delete directories.

The Universal Job Submission Filter JSF

The Perl script *submit.pl* reads the job file constructed by (mini-)SMS, inserts headers and trailers and, where needed, modifies assignments to environment variables. It then submits the thus constructed job.

The information which headers and trailers to add, and which environment variables to change, is kept in a database, *scripts/submission.db*. For the ease of parsing, this database is in fact a file in Perl syntax, executed by the filter *submit.pl*.

In this data base the user specifies on which host to run the job. The database contains, where appropriate, the LoadLeveler or PBS (or any other queuing system) directives to be inserted in the header. If the job runs under mini-SMS, it also inserts the mini-SMS instructions to communicate with mini-SMS. The headers can also be used to insert instructions to read input data from a different host, where needed, and the trailers to send output data to a different host, again where needed. (Currently, no in/output files are transferred because all jobs run on one host.)

Because the submission filter may need changes when the HIRLAM system is ported to a local computer, a more extensive description of it is added as an appendix.

Initializing a run

In the new structure, the experiment working directory is kept on the host on which mini-SMS is run. A run starts with a task, `InitRun`, to create the necessary experiment directories, like `HL_DATA`, on each host defined in `Env_system` (`HOST0` being the local host, other hosts consecutively named `HOST1`, `HOST2`, ...). At ECMWF, `HOST0=ecgate` and `HOST1=hpca-batch`. `InitRun` also copies all scripts, the modifications for this experiment, as kept in `HL_WD`, and the `Env_system` file, to each host. After `InitRun`, no job will access the files in `HL_WD`; instead, the modifications are assumed to be in the copies of `HL_WD` on the host on which the job runs.

`InitRun` also creates a file to be included by (mini-)SMS into all subsequent job scripts. This file is called `hostDescriptions.h`. It is included from `hosts.h`, which itself is included by every job (except `InitRun`). It contains the environment variables for each host; e.g. `HL_DATA1` is `HL_DATA` on host `HOST1`. By including this information in the job to be submitted through the Universal Job Submission Filter, the filter knows those data, and thus can locate job output files, `Env_system`, *etc.*, on the host on which the job is to execute, or on the host from which the job is to read input data. This file also contains the assignments made by the user on the `Hirlam` command line argument, except `DTG`. (e.g. in `'Hirlam start DTG=2003040506 DTGEND=2003040507 LL=48'` it will contain `DTGEND=2003040507 LL=48`), to ensure that these assignments are seen by all jobs.

`InitRun` uses the environment variables `RSH` (default: `rsh`) and `RCP` (default: `rcp`) to access remote hosts. `InitRun` must run on `HOST0`.

Note that as opposed to earlier practice, a change in a script in `HL_WD` will not take effect anymore after `InitRun` has run. In stead, `InitRun` must be re-run to copy the modified script(s) to the proper location on each host.

CollectLogs and LogProgress

Because the `stdout/stderr` files of all jobs are still written on `HOST0`, through NFS, `CollectLogs` still runs on `HOST0`.

The progress log files (`progress.log` and `progressPP.log`) are kept in `HL_WD` on `HOST0`. Hence, the tasks to log the progress are also run on `HOST0`.

Archiving: change directory structure

In the old system, the files to be archived were collected in large `tar` files, and then stored in the root directory of the main experiment archive (`HL_EXP`). In the new system, the files are stored directly, without `tar`, in subdirectories of `HL_EXP`, usually like `yyyy/mm/dd/hh`; but monthly statistics in `yyyy/mm/hh` and climate files in the subdirectory `ClimateFiles`.

The advantage is that retrieval becomes much faster (smaller files, no `un-tar-ring`). The archival system detects much more efficiently whether a certain file is or is not available from the archive. By using subdirectories the total size of a single directory stays limited, even for experiments running over many cycles.

In the old system, if a file had to be retrieved from the archive, the appropriate `tar` file was de-archived and `un-tar-red`. If there was an old `tar` file already in the archive, from an earlier, possibly failed, run of the experiment, it thus could happen that the `un-tar-ring` overwrote files from the new experiment. This was avoided by running a task to remove possibly existing `tar` files from the archive. In the new system, this task is no longer needed.

Acknowledgments

Discussions with and contributions by Ole Vignes and Niko Sokka are gratefully acknowledged. John Greenaway and other ECMWF staff gave a lot of advice.

Appendix

Programming notes on the Universal job submission filter JSF

The filter consists of two Perl scripts: `Submit.pl` contains the main program, and `submission.db` contains the information on how to submit each task.

Submit.pl

This Perl script takes the following actions:

1. Parse the command line arguments. The name of the job file to be filtered is the last argument. Optionally, under the option `-o` the user can give the name of the file the job should write its `stdout/stderr` to.

2. Read the job file to be submitted. From it, extract those environment variables that are specified on lines exactly matching:

```
VARNAME=VALUE export VARNAME
```

The filter will know these variables as Perl variables (in the example, `$VARNAME` would be set to `VALUE`).

3. Extract a few variables from the environment passed to the filter.

4. Read the submission database by executing `submission.db`. If after this the variable `$complete` is set, the filter will not submit any job, but rather send messages to mini-SMS to indicate that the job should be treated as "complete".

5. From `submission.db`, the filter will receive headers and trailers to be added to the job file.

These headers and trailers are edited as follows: strings of the form `%STRING1%` will be replaced if

in the hash %edit there is a value \$edit{STRING1}; this value will be the replacement string. This is similar to actions by (mini-)SMS, but note that there is no recursion here.

6. The job file, with the headers and trailers, will be written to a file with the same name, to which -q (for: to queue) is appended.

7. From submission.db the filter should also have received a string with which to submit the job. This string is in \$submit. This string will also be edited according to the hash %edit, and then it will be sent to the system.

In principle, the filter does not signal anything to (mini-)SMS. Hence, a job will be seen to be active after the submitted job goes into execution; and it will be reported complete or aborted when that job completes or aborts. However, as already indicated above, if the filter is told that the job is complete, it will not submit the job, but rather take over its actions to signal to mini-SMS about the apparent success. Furthermore, even though the filter is fairly robust, it may fail to submit its output for one reason or another (e.g. the disk is full, so it cannot store the -q file). On those occasions, the filter will send the 'aborted' signal to mini-SMS, so that users will be aware of its failure.

submission.db

This Perl script constructs the headers and trailers to each job to be submitted (\$headers and \$trailers, resp.) and the submission sequence (\$submit). The jobs to be submitted can be distinguished at this stage by their full SMS node name (SMSNAME).

In practice, many HIRLAM tasks can be submitted with a common set of headers. E.g., most jobs will only need a CP time allocation of say 300 s, and run on a single processor. To make it easy for you to write those headers, and modify say the required CP time only for a few jobs, the following construct is used: write in the header the string %CP_TIME% where you want the CP time to appear. Prescribe that this string must be replaced by (say) "300" by default by the following statement:

```
$edit{ CP_TIME } = 300
```

but override this for the few bigger jobs.

This explains why submission.db has the following lines, for the parallel jobs at ECMWF:

```
if ( $COMPONENT eq "ECMWF" ) {
  if ( $SMSNAME =~ m~/AnUA/Execute~ and $SCRIPT =~ m~VARan~ ) {
    $edit{ CLASS } = "np";
    $edit{ LL_WALL_CLOCK_LIMIT } = "01:00:00,00:50:00";
    $edit{ LL_NODE } = 1;
    $edit{ LL_TASKS_PER_NODE } = 7;
    $jobclass = "parallel";
  } elsif ( $SMSNAME =~ m~/PpAn/Execute~ and $SCRIPT =~ m~Postpp~ ) {
    $edit{ CLASS } = "np";
    $edit{ LL_WALL_CLOCK_LIMIT } = "00:20:00,00:10:00";
    $edit{ LL_NODE } = 2;
    $edit{ LL_TASKS_PER_NODE } = 8;
    $jobclass = "parallel";
  } elsif ( $SMSNAME =~ m~/Forecast/Execute~ and $SCRIPT =~ m~Prog~ ) {
    $edit{ CLASS } = "np";
    $edit{ LL_WALL_CLOCK_LIMIT } = "02:30:00,02:20:00";
    $edit{ LL_NODE } = 2;
    $edit{ LL_TASKS_PER_NODE } = 8;
    $jobclass = "parallel";
  }
}
```

This specifies job class, wall clock time limits, number of nodes, etc., for the 3DVAR scheme, the postprocessing of the analysis, and the forecast model, resp.

Similar statements apply for the trailers and the submission string `$submit`.

If a job is to be run on a different host than `HOST0` (`HOST0` must be the node running (mini-)SMS), specify its node number in `$host`. If it is to be run in the background (implying it runs on `HOST0`), specify `$bckgrnd = 1`.

At this stage, you can modify the `export` statements in the job to be submitted: if you modify a variable that was read from the job input file from a line like:

```
VARNAME=VALUE export VARNAME
```

then the filter will replace `VALUE` in this line by the new value. With this mechanism, it is possible to modify the environment of the submitted job.

In particular, this mechanism is used to get the full pathnames of the system description file (`Env_system`) and the pseudo-environment file (`SETENV`), on the target host.

For jobs that do not have to be submitted at all, you can set `$complete` to 1. In this way you will be able to avoid that a job to construct a certain file is run if the file already exists. This is particularly useful if the job would have to queue a long time to get the required resources, and then after all that waiting would do nothing when it finds that the target file already exists. To accomplish optimizations like this, though, you will have to dive deeply into Perl programming.