Schedule WRF model executions in parallel computing environments using Python

A.M. Guerrero-Higueras, E. García-Ortega and J.L. Sánchez

Atmospheric Physics Group, University of León, León, Spain

J. Lorenzana

Foundation of Supercomputing Center of Castile and León, León, Spain

V. Matellán

Dpt. Mechanical, IT, and Aerospace Engineering, University of León, León, Spain

ABSTRACT

The Weather Research and Forecasting (WRF) Model is a next-generation mesoscale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs. WRF is suitable for a broad spectrum of applications across scales ranging from meters to thousands of kilometers. Preparing an operational forecast that uses available resources efficiently for its execution requires certain programming knowledge in parallel computing environments. This work shows how to construct a tool that allows for scheduling operational executions of the WRF model. This tool will allow users without advanced programming knowledge to perform their own periodic planning operative executions of WRF model.

1. Introduction

The Weather Research and Forecasting model (WRF), Skamarock et al. (2005), is an non-hydrostatic atmospheric simulation model for a limited area, sensitive to the characteristics of the terrain, and designed to forecast atmospheric circulation on synoptic, mesoscale, and regional scales. It was developed in collaboration with the National Oceanic and Atmospheric Administration (NOAA), the National Centers for Atmospheric Research (NCAR), and other organizations.

The implementation of the model is prepared to work in parallel computing environments with shared memory, using OpenMP, and with distributed memory, using MPI. Additionally, the model has the capacity to combine both technologies.

The WRF model is composed of a series of modules, UCAR (2012). Each module corresponds to a different function: GEOGRID allows for the configuration of the geographic area of study. UNGRIB prepares the data for the initialization of the model, which is normally established by the output of another model with greater spatial coverage. METGRID prepares the boundary conditions, adapting itself to the characteristics of the domains defined in GEOGRID. REAL does a vertical interpolation from the pressure levels to a system of normalized Sigma coordinates. The WRF module solves the physical forecasting and diagnostic equations that allow for a forecast with a predetermined temporal horizon.

The WRF model is designed so that each module must be executed independently. This provides many advantages, especially if it is only used for atmospheric research. However, if the objective is to plan its execution in order to get operational forecasts, it can be inconvenient. The Atmospheric Physics Group (GFA) at the University of León (ULE), uses the model for operational forecasting.

It is important to point out that there are no tools that allow for scheduling executions. This makes centers that want the model for operational use define their own solutions ad hoc. In order to do so, it might be necessary to have at least some basic knowledge of programming, especially if the objective is to make use of its capacity to run parallel computations.

The GFA has implemented a tool, known as PyWRFscheduler[1], using Python, which resolves this problem and that can be configured by any researcher without previous programming knowledge. This tool guarantees the optimization of resources in a parallel computing environment. This can be very important, since the access to this type of environment is usually limited, and often, it can incur high economic costs.

Normally, access to these environments is controlled by a job scheduler, who is usually in charge of sending jobs

and reserving the necessary nodes to run them in a parallel computing cluster. The GFA carries out its forecasts in a cluster from the Foundation of Supercomputing Center of Castile and León (FCSCL), using up to 360 nodes in some cases. The FCSCL uses *Sun Grid Engine* (SGE), Sun Microsystems (2002), to manage sending their jobs to *Caléndula*, its parallel computing cluster.

The most common way—which is also the least efficient—of executing the model in a parallel computing environment is to send only one job to the job scheduling. This does not guarantee a complete optimization of resources, since not all of the modules of the WRF model are prepared to be executed in a parallel computing environment. Usually, only the REAL and WRF models are done this way, since executing both of these modules implies the largest part of overall time that the model is run. As such, it is necessary to point out that during the execution of other modules, only one of the reserved nodes is used. This means that during a period of time, some nodes in the cluster are reserved but unoccupied. This period of time can be relatively large, especially if post-processing of the output is carried out.

The tool developed by the GFA optimizes the use of parallel resources. Thus, instead of sending one job to the job scheduler, a job for each module is sent, plus an extra job for the post-processing of output. In this way, it is possible to indicate that the job scheduler should only reserve the exact number of nodes needed in order to carry out each job, which presents important advantages, as previously shown.

2. Prior Work

As seen previously, the WRF model is composed of various modules that have to be executed sequentially in order to obtain a weather forecast. Each module has different computation needs. *GEOGRID*, *UNGRIB*, and *METGRID* are habitually executed in one node, while REAL and WRF are normally executed in various parallel nodes.

Writing a shell script that makes it possible to execute each module sequentially, and later send it to a parallel calculation environment as one job, is simple. `wrf.sh` script, which is available on line at GFA website[2], shows an example ready to be sent to the SGE job scheduler of the parallel calculation cluster of the FCSCL. In this example, 360 nodes are reserved to execute the job, however, the execution of the WRF model alone makes use of all of the nodes. During execution of the rest of the modules, there are nodes that do not perform any calculations.

In Table 1, we can see the execution time of each module. During the execution of *GEOGRID*, *UNGRIB*, *METGRID*, and, on a lesser scale, during the execution of *REAL* module, there are nodes that are not in use. We have to add this accumulated time to the time of each download of entry data not included in the table, and the time of post-processing of the output if and when it is carried out.

Table 1. Execution time of different modules of the WRF model in 360 nodes.

Module	Running time	Reserved nodes	Used nodes	Unused nodes
UNGRIB	$13m12s$	360	1	359
METGRID	$10m48s$	360	1	359
REAL	$1m20s$	360	32	328
WRF	$1h27m11s$	360	360	0

This work method implies a waste of available resources, which makes the GFA consider a more efficient solution.

3. Software Architecture

In order to solve the problem presented in the previous section, the GFA has designed a tool, called PyWRFscheduler, which sends different jobs to a SGE job scheduler in sequence, following a *work-flow* of previously defined tasks.

The first step consists of discomposing the execution of the model in independent tasks. Each task is implemented in an independent Python script, all of them are available on line at GFA website:

- `preprocess.py`[3]: This script processes tasks before the model execution.
- `geogrid.py`[4]: Responsible for executing the *GEOGRID* module of the WRF model.
- `ungrib.py`[5]: Responsible for executing the *UNGRIB* module.
- `metgrid.py`[6]: Responsible for executing the *METGRID* module.
- `real.py`[7]: Responsible for executing the *REAL* module.
- `wrf.py`[8]: Responsible for executing the *WRF* module.

The output that the WRF model produces is in netCDF format, Unidata (2012). Habitually, graphic representations are generated using output in order to visualize results. These graphics can be generated by using an additional script that can be included in the tasks *work-flow*.

[2] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/wrf.sh

[3] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/preprocess.py
[4] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/geogrid.py
[5] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/ungrib.py
[6] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/metgrid.py
[7] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/real.py
[8] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/wrf.py

The implementation of these types of scripts can be sequential, since they do not require much additional computational work. However, if the number of graphics is very high, the execution time increases, and it is interesting to think about a parallel solution in these cases, Guerrero-Higueras et al. (2012).

In addition to the previous scripts, an additional script is necessary. This process will be responsible for sending the tasks to the job scheduler in the proper order. It should be executed in a node from which jobs can be sent to a parallel environment. This process is implemented with another Python script: `wfmanager.py`[9].

Figure 1 shows a diagram with the interaction of each of the scripts with the job scheduler and with a log file that the job scheduler generates.

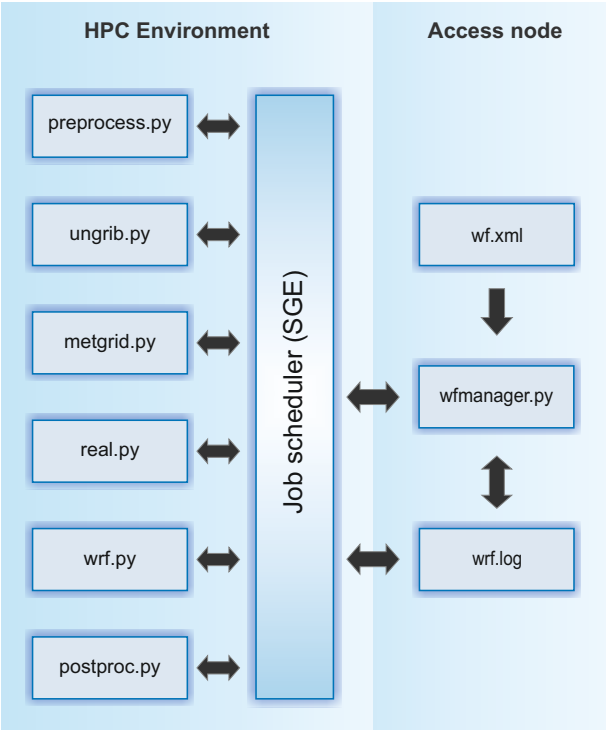

FIG. 1. Software Architecture.

The following sections explain the function of each script.

a. `wfmanager.py` *script*

This script is responsible for coordinating the entire sequential sending process of tasks. In order to monitor the beginning and end of each task, wfmanager.py uses the log file that the SGE job scheduler generates with the result of the execution of each job.

The first step consists of defining a work-flow that includes all of the tasks. In order to define this work-flow, a file in XML format is used, W3C (1996). This XML file follows a few rules and contains a series of entities:

- A `work-flow` entity with the definition of the same *work-flow*.
- *N* elements with the entity `task`, one for each of the tasks that compose the *work-flow*.

Figure 2 represents the hierarchical structure of the entities that form the work-flow. In the following sections, all of the entities that define the work-flow are detailed, along with the different attributes that they support.

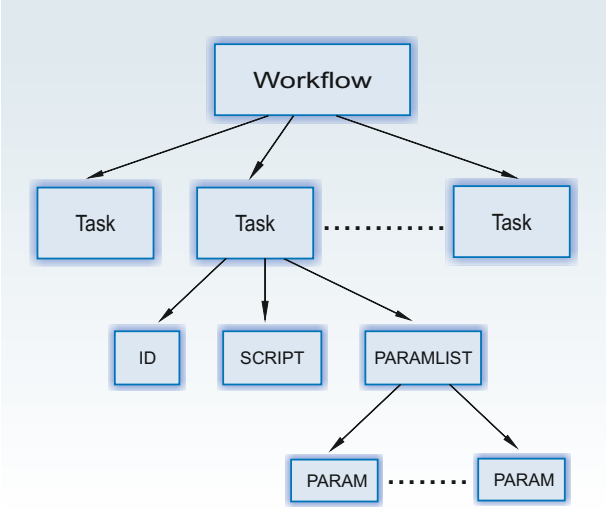

FIG. 2. *Work-flow* hierarchical structure.

1) `work-flow` ENTITY

The `work-flow` entity should contain only one series of `task` entities that define each task. `work-flow` supports two attributes. The `date` attribute contains the date and forecast start-time, and the `forecast` attribute that indicates the number of forecast hours from the start time.

2) `task` ENTITY

The `task` entity contains the definition of the task. A sequence of elements in this entity defines the `work-flow` entity. Each `task` entity should contain an element for each one of the following entities:

- `id` entity: Assigns a name for a task. It supports the following attributes:
  - `continueTag`: This will be the value that the *work-flow* manager looks for, the `wfmanager.py`

[9] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/wfmanager.py

script, in the log file created by the SGE job scheduler, in order to make sure that the task is executed correctly and that it is possible to continue with the the next one.

  - `abortTag`: This will be the value that the workflow manager looks for, `wfmanager.py`, in the log file created by the SGE job scheduler, in order to make sure that if an error has occurred when performing the task, and it is not possible to continue with the next one.

- `script` entity: This indicates the script path that the task executes. It supports the following attributes:

  - `nodes`: Specifies the number of nodes in the parallel calculation environment where the task is executed.

  - `queue`: Specifies the execution queue in the parallel calculation environment where the task is executed.

- `paramlist` entity: Contains the list of parameters that each script needs to carry out a task. Each script contains a different number of parameters. A `param` entity is defined for each parameter. Each `param` element has a value for each parameter. `param` supports the `name` attribute that is assigned to each parameter.

`wf.xml` file shows, available on line at GFA website[10] under the terms of the GNU General Public License, the definition of a work-flow.

Once the *work-flow* file is defined in the XML file, a `wfmanager.py` script is executed. `wfmanager.py` covers the XML file using a parsing process, implemented using the API DOM (Document Object Model API) from Python, Python Software Foundation (1990). For each `task` entity that appears in the *work-flow*, `wfmanager.py` sends a job to the SGE job scheduler. In order to send jobs to the SGE job scheduler, `wfmanager.py` uses the `qsub` command, along with a series of arguments and the script path:

- Working directory (`-wd`): base directory where the script works.

- Parallel environment (`-pe`): number of nodes needed to be reserved to execute each job.

- Name (`-N`): identifier of job.

- Output (`-o`): file path that is required to redirect the standard output.

- (`-j`): Specifies whether or not the standard error stream of the job is merged into the standard output stream.

- Shell (`-S`): *Shell* where the script is executed.

- Queue (`-q`): queue where the job is sent.

`wfmanager.py` uses values for the attributes and entities contained in the `task` entity in order to select the script and assign values to the `qsub` arguments.

b. `preprocess.py` *script*

This script does tasks before the execution of the model:

- It is responsible for downloading information in order to initiate the WRF model. In the GFA executions, the WRF model initiates with the information from the Global Forecasting[11] (GFS) model.

- It updates data in the model's configuration files. Specifically, it modifies the date and time of the forecast in the `namelist.wps` and `namelist.input` files, as shown in Figures 3 and 4.

```
&share
 start_date = '2013-01-22_00:00:00',
              '2013-01-22_00:00:00',
              '2013-01-22_00:00:00',
 end_date   = '2013-01-26_00:00:00',
              '2013-01-24_00:00:00',
              '2013-01-24_00:00:00',
...
```

FIG. 3. Time control section in `namelist.wps` file.

If both tasks are executed correctly, `preprocess.py` writes the `continueTag` defined for the `task` in the log file, otherwise, `preprocess.py` writes the `abortTag`.

c. `ungrib.py` *script*

This script is responsible for the execution of the *UNGRIB* module. First, it should connect information that allows for the initialization of the model so that *UNGRIB* can use it. The same WRF model provides a script with a shell that performs this function: `link_grib.csh`. It is only necessary to indicate the path and the pattern for the files that contain the information needed to initialize the model. After connecting the entry data, one only has to execute the `ungrib.exe` program included in the model's software.

If `ungrib.exe` is executed correctly, `ungrib.py` writes the `continueTag` defined for the `task` in the log file, otherwise, `ungrib.py` writes the `abortTag`.

[10] http://gfa.unileon.es/data/toolbox/PyWRFscheduler/wf.xml

[11] Carried out by the National Oceanic and Atmospheric Administration (NOAA).

```
&time_control
 start_year   = 2013,2013,2013,
 start_month  = 01,01,01,
 start_day    = 22,22,22,
 start_hour   = 00,00,00,
 start_minute = 00,00,00,
 start_second = 00,00,00,
 end_year     = 2013,2013,2013,
 end_month    = 01,01,01,
 end_day      = 26,24,24,
 end_hour     = 00,00,00,
 end_minute   = 00,00,00,
 end_second   = 00,00,00,
...
```

FIG. 4. Time control section in `namelist.input` file.

d. `metgrid.py` *script*

The `metgrid.py` script is responsible for the execution of the *METGRID* module. It uses the information obtained from the *UNGRIB* module, and, as such, has to be executed afterwards. One only has to execute the `metgrid.exe` program.

If `metgrid.exe` is executed correctly, `metgrid.py` writes the `continueTag` defined for the `task` in the log file, otherwise, `metgrid.py` writes the `abortTag`.

e. `real.py` *script*

This script is responsible for the execution of the *REAL* module of the WRF model. It uses information obtained from the *METGRID* module, and, as such, has to be executed afterwards. One only has to execute the `real.exe` program using `mpirun`, indicating the number of nodes.

If `real.exe` is executed correctly, `real.py` writes the `continueTag` defined for the `task` in the log file, otherwise, `real.py` writes the `abortTag`.

f. `wrf.py` *script*

The `wrf.py` script is responsible for the execution of the *WRF* module. It uses information obtained from *REAL* module, and, as such, has to be executed afterwards. One only has to execute the `wrf.exe` program, using the `mpirun`, indicating the number of nodes.

If `wrf.exe` is executed correctly, `wrf.py` writes the `continueTag` defined for the `task` in the log file, otherwise, `wrf.py` writes the `abortTag`.

4. Development Process

The PyWRFscheduler tool, Copyright 2013 Atmospheric Physics Group, was developed using Python. All of the scripts were tested in the FCSCL cluster. They are available to the scientific community, under the terms of the GNU Lesser General Public License[12][13] (LGPL). They are available on line at the GFA website[14].

5. Application: GFA's operational forecasts

Operational forecasts are available on line at GFA website. Two types of forecasts are calculated, over three added domains with different spatial resolutions, 27, 9, and 3 km, respectively, that include different geographical areas in the southeast of Europe, as seen in Figures 5 and 6.

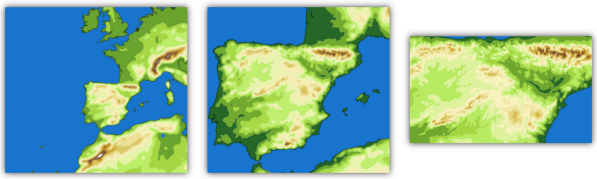

FIG. 5. Domains 1, 2, and 3 for *Type-I* forecasts.

FIG. 6. Domains 1, 2, and 3 for *Type-II* forecasts.

- *Type-I* Forecasts: Calculated twice per day, beginning at 0:00 UTC and at 12:00 UTC, with a temporal horizon of 96 hours for the first domain and 48 for the other domains.
- *Type-II* Forecasts: Calculated twice per day, beginning at 6:00 UTC and at 18:00 UTC, with a temporal horizon of 48 hours.

Tables 2 and 3 show the average time of execution and the nodes used for the different modules of the WRF model, in *Type-I* and *Type-II* forecasts , respectively, sending only one job to the SGE job scheduler.

Tables 4 and 5 show the average time of execution of the nodes used for the different modules of the WRF model for *Type-I* and *Type-II* forecasts using the PyWRFscheduler tool, respectively.

[12] http://www.gnu.org/licenses/gpl.txt
[13] http://www.gnu.org/licenses/lgpl.txt
[14] http://gfa.unileon.es/?q=toolbox

TABLE 2. Execution of *Type-I* forecasts, sending only one job to the SGE job scheduler.

Module	Running time	Reserved nodes	Used nodes	Unused nodes
UNGRIB	$7m22s$	128	1	127
METGRID	$2m10s$	128	1	127
REAL	$2m3s$	128	16	112
WRF	$45m35s$	128	128	0

TABLE 3. Execution of *Type-II* forecasts, sending only one job to the SGE job scheduler.

Module	Running time	Reserved nodes	Used nodes	Unused nodes
UNGRIB	$8m58s$	288	1	287
METGRID	$7m44s$	288	1	287
REAL	$1m45s$	288	32	256
WRF	$1h38m15s$	288	288	0

6. Conclusion

By executing the WRF model, and sending only one job to the computing environment, time intervals were observed in which there were unused nodes, as seen in Tables 2 and 3. By using the PyWRFscheduler, which sends jobs to the computing environment for each module, this did not occur, since only nodes that were necessary were reserved for each case.

After doing tests, it is important to point out that the use of the PyWRFscheduler tool to execute the WRF model, guarantees optimal use of the resources in a parallel computing environment. Thus, its use suggests only a small upcharge in the total time of execution.

Acknowledgments.

The authors would like to thank the Junta of Castile and León for their economic support via the LE176A11-2 project. This study was supported by the following grants: CGL2010-15930; MICROMETEO (IPT-310000-2010-22).

TABLE 4. Execution of *Type-I* forecasts using PyWRF-scheduler.

Module	Running time	Reserved nodes	Used nodes	Unused nodes
UNGRIB	$7m36s$	1	1	0
METGRID	$2m19s$	1	1	0
REAL	$2m33s$	16	16	0
WRF	$46m05s$	128	128	0

TABLE 5. Execution of *Type-II* forecasts using PyWRF-scheduler.

Module	Running time	Reserved nodes	Used nodes	Unused nodes
UNGRIB	$8m59s$	1	1	0
METGRID	$8m14s$	1	1	0
REAL	$1m53s$	32	32	0
WRF	$1h38m32s$	288	288	0

REFERENCES

Guerrero-Higueras, A., E. García-Ortega, V. Matellán-Olivera, and J. Sánchez, 2012: Procesamiento paralelo de los pronósticos meteorológicos del modelo WRF mediante NCL. *Actas de las XXIII Jornadas de Paralelismo JP2012, ISBN: 978-84-695-4471-6.*, 55–60.

Python Software Foundation, 1990: The document object model API. `http://docs.python.org/2/library/xml.dom.html`.

Skamarock, W., J. Klemp, J. Dudhia, D. Gill, D. Barker, W. Wang, and J. Powers, 2005: A description of the advanced research wrf version 2. *NCAR Tech. Note NCAR/TN-468+STR.*

Sun Microsystems, 2002: Sun^{TM} ONE Grid Engine Administration and User's Guide.

UCAR, 2012: User's guide for the advanced research WRF (ARW) modeling system. `http://www.mmm.ucar.edu/wrf/users/docs/user_guide_V3/contents.html`.

Unidata, 2012: NetCDF users guide. `https://www.unidata.ucar.edu/software/netcdf/docs/user_guide.html`.

W3C, 1996: Extensible markup language (xml). `http://www.w3.org/XML/`.