Scripts for Restart Files

I show you how to use Python for compressing MPI-ESM restarts. Usually, you only need to think about the experiments that you would want to restart in the future and what dates you want to store. Despite this selection, you will store all the restart files from all the submodels. If you do not do that, you will not be able to use them.

Python script

What do I want to store?

The restart files contain the state of the system at a given point in time. You use them to start again a simulation from a given point. For example, when you ended your run at a certain year and you want to extend the run, you take the restart files and initialise the model with the latest state. Moreover, when you run MPI-ESM normally, it runs for one year, stops, and restarts for next year until you reach the last year prescribed in your run script. Thus, you must make sure that you store the restart files of all the submodels for your restart date. Then here the approach differs slightly from the output example. Here we do not need the substream level and we pack everything of a stream in a single file.

As with the scripts for output, the script that I present is extensible and straightforward. First, we need to import the following modules

```
import os
import subprocess as sp
from dask.distributed import Client
```

The `os` module allows us to perform some operative and file system tasks, such as directory changing. The `subprocess` module makes possible to launch subprocesses to run programs outside the python session, in particular, shell programs. The `dask.distributed` module enables the use of MISTRAL to distribute and parallelise computations. About this last point, the script is a direct application of the Reserve and Use method.

Once we have imported the modules, we first connect to the computing resources of MISTRAL (section 0 of the code). In the following sections, I will talk about it. Thus, for now, let us give it for granted.

```
# 0. Acquire the reserved resources in the cluster

client=Client(scheduler_file=('/scratch/x/x000000/dask-work/Restarts'+
                              '/scheduler_Restarts.json'))
print(client)
```

With the resources secured, we now define the paths to the model output and to the temporary space where we keep the compressed files (section I). I provide dummy paths. I strongly recommend that the temporary space is in `/scratch`, because of the large amounts of disk space that the compressed files occupy.

```
# I. Definition of paths to the data and to the scratch space

exps_path=os.path.abspath(("/work/yy0000/x000000/"+
                           "mpiesm-1.2.01p1-fixvar/experiments"))+"/"
outs_path=os.path.abspath("/scratch/x/x000000")+"/foobar/experiments/"
```

Now, in section II, we set some variables that will describe which experiments we will process and the submodels (streams). The assumption here is that the experiments directory has the following structure: `<experiment>/restart/<stream>/`. `exps` is the list of experiment names. `strms` define the submodel list (`oasis3mct` is the coupler of all the submodels).

```
# II. Definition of experiments paths

exps=["djc_cntrl_stnd"] # Experiment names
for i in range(7):
 exps.append("djc_cldfc.{0}_stnd".format(i))

strms=["echam6",
       "hamocc",
       "jsbach",
       "mpiom",
       "oasis3mct"
      ] # Streams of restarts

# III. Strings with evaluable fields used in helper functions

path="{0}{1}/{2}"
file="{0}_{1}.tar.gz"
```

In section III, we define two strings with evaluable fields that we will use in the helper function, which is section IV of the code. The helper function is the core of the script. It is the function that will process the files. That is why I called it `tar_and_zip_restarts`. It has four inputs, the path to the experiments, the path to the compressed restarts, the name of the experiment, and the name of the stream to be processed. The first step is to construct the string that contains the path to the `restart` directory where we store the data, as well as the `restart` directory where the compressed files will be stored. Then, we make the latter directory if it does not exist yet. The third step is to add the file name to the path of the compressed output with the following structure: `<output_path>/<experiment>/restart/<stream>_restarts.tar.gz`. The fourth step begins with changing to the experiment output directory. After that operation we apply a `tar` to them and pipe the result to a `pigz` command to compress. The function returns, after a successful compression, the path to the output. Here I will store all the restart files of my experiments. Then you can use `find` or similar to filter out the results and select a subset of restarts based on the date. *However, if you will only store a single date, I strongly recommend that you do all the process manually.*

```
# IV. Helper function
```

```
def tar_and_zip_restarts(experiments_path,
                         output_path,
                         experiment,
                         stream
                        ):

 # 1 Path construction

 path_to_files=path.format(experiments_path,experiment,"restart")
 path_to_output=path.format(output_path,experiment,"restart")

 # 2 Construct output directory if missing

 command=["mkdir","-p",path_to_output] # Command
 sp.run(command,check=True) # Execute the command

 # 3 Add file name to the output path

 path_to_output+="/"
 path_to_output+=file.format(stream,"restarts")

 # 4 concatenate and compress all files in the output file

 os.chdir(path_to_files) # Change to the input directory
 with open(path_to_output,"w") as out: # With the output file opened...
  c1=["tar","cvf","-",stream] # Tar command
  c2=["pigz"] # Parallel compression command
  r1=sp.Popen(c1,stdout=sp.PIPE) # Execute tar and pipe the output...
  r2=sp.Popen(c2,stdin=r1.stdout,stdout=out) # ...to the pigz command
  r1.stdout.close() # When pigz ends, close stdout of tar
  r3=r2.communicate()[0] # Check whether pigz have ended or not

 # 5 Return the full path to the output

 return path_to_output
```

How do I want to store it using MISTRAL power?

The next step, as said in the scripts for output, depends heavily on i) the MISTRAL resources that your group have and ii) How fast do you want to have the result. In the example that I provide, I process concurrently all the streams of each experiment and I wait until it finishes with the experiment to process the next experiment. That is why in contrast to the example for output, the result gathering phase is within the for-loop.

```
# V. Parallel compilation of files

# In this case the settings of the workers are for five nodes. One node for
# each stream. We parallelise at the level of streams. We gather the results
# in the same for cycle, which means that it will wait until all the streams
```

```
# of a given experiment are processed to proceed to the next experiment.

list_tasks={} # Dictionary of tasks
list_results={} # Dictionary of results
for exp in exps: # Cycle throughout experiments...
 print("Storing restarts of experiment {}...".format(exp))
 list_tasks[exp]=[ client.submit(tar_and_zip_restarts,
                                 exps_path,
                                 outs_path,
                                 exp,
                                 strm,
                                 pure=False
                                )
                   for strm in strms
                 ] # Submit to the cluster for each stream
 list_results[exp]=[ i.result() for i in list_tasks[exp] ] # Wait
```

The last two sections of the code (VI and VII) are only for closing the connection to the MISTRAL resources and cancelling the processes.

```
# VI. Relinquish control of cluster reserved resources

client.close()
print(client)

# VII. Cancel the processes in the cluster

command=["scancel","-u","x000000","--name","dask-workers-Restart"]
sp.run(command,check=True)
command=["scancel","-u","x000000","--name","dask-scheduler-Restart"]
sp.run(command,check=True)
```

The complete script

Restarts.py

```
import os
import subprocess as sp
from dask.distributed import Client

# 0. Acquire the reserved resources in the cluster

client=Client(scheduler_file=('/scratch/x/x000000/dask-work/Restarts'+
                              '/scheduler_Restarts.json'))
print(client)

# I. Definition of paths to the data and to the scratch space

exps_path=os.path.abspath(("/work/yy0000/x000000/"+
```

```
                                "mpiesm-1.2.01p1-fixvar/experiments"))+"/"
outs_path=os.path.abspath("/scratch/x/x000000")+"/foobar/experiments/"

# II. Definition of experiments paths

exps=["djc_cntrl_stnd"] # Experiment names
for i in range(7):
 exps.append("djc_cldfc.{0}_stnd".format(i))

strms=["echam6",
       "hamocc",
       "jsbach",
       "mpiom",
       "oasis3mct"
      ] # Streams of restarts

# III. Strings with evaluable fields used in helper functions

path="{0}{1}/{2}"
file="{0}_{1}.tar.gz"

# IV. Helper function

def tar_and_zip_restarts(experiments_path,
                         output_path,
                         experiment,
                         stream
                        ):

 # 1 Path construction

 path_to_files=path.format(experiments_path,experiment,"restart")
 path_to_output=path.format(output_path,experiment,"restart")

 # 2 Construct output directory if missing

 command=["mkdir","-p",path_to_output] # Command
 sp.run(command,check=True) # Execute the command

 # 3 Add file name to the output path

 path_to_output+="/"
 path_to_output+=file.format(stream,"restarts")

 # 4 concatenate and compress all files in the output file

 os.chdir(path_to_files) # Change to the input directory
 with open(path_to_output,"w") as out: # With the output file opened...
  c1=["tar","cvf","-",stream] # Tar command
  c2=["pigz"] # Parallel compression command
  r1=sp.Popen(c1,stdout=sp.PIPE) # Execute tar and pipe the output...
```

```
  r2=sp.Popen(c2,stdin=r1.stdout,stdout=out) # ...to the pigz command
  r1.stdout.close() # When pigz ends, close stdout of tar
  r3=r2.communicate()[0] # Check whether pigz have ended or not

 # 5 Return the full path to to the output

 return path_to_output

# V. Parallel compilation of files

# In this case the settings of the workers are for five nodes. One node
for
# each stream. We parallelise at the level of streams. We gather the
results
# in the same for cycle, which means that it will wait until all the
streams
# of a given experiment are processed to proceed to the next
experiment.

list_tasks={} # Dictionary of tasks
list_results={} # Dictionary of results
for exp in exps: # Cycle throughout experiments...
 print("Storing restarts of experiment {}...".format(exp))
 list_tasks[exp]=[ client.submit(tar_and_zip_restarts,
                                 exps_path,
                                 outs_path,
                                 exp,
                                 strm,
                                 pure=False
                                )
                   for strm in strms
                 ] # Submit to the cluster for each stream
 list_results[exp]=[ i.result() for i in list_tasks[exp] ] # Wait

# VI. Relinquish control of cluster reserved resources

client.close() # Close the connection to the scheduler
print(client) # Print to check if it was disconnected

# VII. Cancel the processes in the cluster

command=["scancel","-u","x000000","--name","dask-workers-Restarts"]
sp.run(command,check=True) # First kill workers
command=["scancel","-u","x000000","--name","dask-scheduler-Restarts"]
sp.run(command,check=True) # Then the scheduler
```

Slurm scripts

To reserve the resources, you need to follow the steps described in the Reserve and Use method. In this case, I reserved one node in the `shared` partition for the scheduler. I also reserve five nodes in the `compute2` partition. In both cases, I only allow one task to run in each node. The scheduler node manages the five workers. Then, when I submit the requests in section V of the Python code, I am sending the instructions to the workers. You should look at the Reserve and Use method. In the following, I provide the scripts to reserve the nodes.

I recommend that you have a specific dask-work directory (possibly in scratch) as a place to store all the log files of the `dask.distributed` operations. The directory should also have several subdirectories for each dask processing task. In case there is a problem, you will debug it better and faster.

launch_scheduler_Restarts.sh

```
#!/bin/bash
#SBATCH --account=yy0000
#SBATCH --job-name=dask-scheduler-Restarts
#SBATCH --workdir=/scratch/x/x000000/dask-work/Restarts
#SBATCH --output=/scratch/x/x000000/dask-work/Restarts/LOG_dask.%j.o
#SBATCH --error=/scratch/x/x000000/dask-work/Restarts/LOG_dask.%j.o
#SBATCH --time=08:00:00
#SBATCH --partition=shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1

rm -rf worker-*
rm *.o
rm scheduler*.json
rm global.lock
rm purge.lock

module purge
module load pigz
export PATH=/work/yy0000/x000000/miniconda3/bin:$PATH
conda activate glamdring

srun dask-scheduler --scheduler-file /scratch/x/x000000/dask-
work/Restarts/scheduler_Restarts.json
```

launch_workers_Restarts.sh

```
#!/bin/bash
#SBATCH --account=yy0000
#SBATCH --job-name=dask-workers-Restarts
#SBATCH --workdir=/scratch/x/x000000/dask-work/Restarts
#SBATCH --output=/scratch/x/x000000/dask-work/Restarts/LOG_dask.%j.o
#SBATCH --error=/scratch/x/x000000/dask-work/Restarts/LOG_dask.%j.o
#SBATCH --time=08:00:00
#SBATCH --partition=compute2
#SBATCH --nodes=16
```

```
#SBATCH --ntasks-per-node=1

module purge
module load pigz
export PATH=/work/yy0000/x000000/miniconda3/bin:$PATH
conda activate glamdring

srun dask-worker --scheduler-file /scratch/x/x000000/dask-
work/Restarts/scheduler_Restarts.json
```

How to run these scripts?

The first thing is that you would want to first use `screen` (or `tmux`) to create a shell session that you can detach without killing your long-lived processing. Inside your `screen` session, you submit to Slurm your request for the scheduler node.

```
sbatch launch_scheduler_Restarts.sh
```

After some time it should be online, then you run the request for workers.

```
sbatch launch_workers_Restarts.sh
```

Once the workers are online, they will connect to the scheduler. Then you can run your script

```
python Restarts.py
```

Remark: In the Slurm scripts there are lines for putting in the search path a `miniconda` distribution and initialising an environment called `glamdring`. If you are using the anaconda3 from the system, just delete those lines and use `module` command to load anaconda3. If you are using your own `miniconda` distribution, you should change those lines and load the proper environment.