Optimising workflow lifecycle management: development, HPC-ready containers deployment and reproducibility

Hands-on guide and Questionnaire

In this hands-on, we are using an adapted version of a workflow used in CAELESTIS for mechanical testing simulation which was developed by Riccardo Cecco (previous member of the Workflows and Distributed Computing Group, BSC) with the guidance of Gerard Guillamet (Dual Technologies Research Group, BSC) and Aravind Sasikumar (AMADE Research UDG). The workflow is distributed with a singularity container described by Fernando Vazquez (Workflows and Distributed Computing Group, BSC) and created with the Image Creation Service developed by Jorge Ejarque (previous member of the Workflows and Distributed Computing Group, BSC) at the eFlows4HPC project. This link[1] provides the instructions about how to create a container with the Container Image Creation tool for an HPC cluster.

The source of the workflow can be found in this link[2]. The main code of the workflow is implemented in *workflow.py*. The workflow has 2 parameters as input:

- a yaml file that configures the workflow indicating the problem to solve (variables to sample, mesh of the object to test and AI models and parameters to analyse)
- the execution directory.

The workflow has three parts:

1. A sampling part which receives the problem definition and generates the Design of Experiments (DoE) matrix (sampling.py file)
2. The simulation part that for each row of the DoE matrix launches a simulation (simulation.py file)
3. An AI part which performs a model selection using the Grid Search algorithm from the dislib and the Decision Tree Regressor from sk-learn (ai.py file). The source code of the Grid Search can be found here and the internal PyCOMPSs tasks (fit_sklearn_estimator and score_sklearn_estimator) that are invoked can be seen here.

The other files contain auxiliary functions that are not relevant for the hands-on.

---

[1] https://github.com/eflows4hpc/image_creation/blob/main/README.md#instructions-to-run-container-image-creation-on-your-local-computer

[2] https://github.com/eflows4hpc/workflow-registry/tree/main/tutorial/HPC_AI_training/src

1. Login to MareNostrum 5 using the username and password provided[3]. Nodes to login to the general purpose partition are:

```
glogin1.bsc.es
glogin2.bsc.es
```

Follow the instructions below to copy and run the workflow in your user space:

2. Copy the workflow in your home directory

```
$ cp -R /gpfs/scratch/nct_312/Tutorial_SC24/CAELESTIS $HOME
```

3. Load the Singularity and COMPSs environment

```
$ cd CAELESTIS
$ CAELESTIS > source load_compss.sh
```

4. We will use the *run.sh* PyCOMPSs script to submit the execution of the workflow to MareNostrum 5. This script has two input arguments: the configuration yaml and the number of nodes. The PyCOMPSs script contains some flags to execute the workflow with containers (*--container*) and the *--graph* flag to generate the task-dependency graph of the execution. Also note, that we are adding the --env_script flag which specifies a script with some environment variables used at execution. Inspect both files to understand the execution.

```
$ CAELESTIS > cat run.sh
$ CAELESTIS > cat app_env.sh
```

5. Run the following command which will execute the workflow generating 8 samples and using 2 computational nodes.

```
$ CAELESTIS > sh run.sh test.yaml 2
```

The execution of the workflow will take about 10 minutes. During this time look at the source code and the test.yaml and try to answer the following questions.

[3] More information: https://www.bsc.es/supportkc/docs/MareNostrum5/logins

Question 1: Identify the different tasks. How many executions of each task will be performed (tasks can be identified in the Python files of the workflow source code and in the *dislib* code; see links in the introduction)?

Question 2: Draw a schema of the task-graph that you consider that will be created

Once the execution is finished, check that the execution has finished without errors and follow the instructions to generate and visualise the graph, where *job_id* is the identifier of the executed job.

```
$ CAELESTIS > pycompss gengraph \
          .COMPSs/<job_id>/monitor/complete_graph.dot
```

Copy the file complete_graph.pdf to your system[4], by executing the following command from a command line in your laptop, where "username" is your MareNostrum5 username:

```
$ my_laptop > scp \
<username>@transfer1.bsc.es:./CAELESTIS/.COMPSs/<job_id>/monitor/complet
e_graph.pdf .

Open the file with your favourite pdf viewer.
```

[4] You can try opening with "`gv .COMPSs/<job_id>/monitor/complete_graph.pdf`" but being a remote connection it will take a while to open the file

Question 3: Is your graph similar to the one generated? If not, what was the reason for your confusion?

Finally, you can play with different configurations modifying the *app_env.sh* script. You can modify the number of processes dedicated to the execution modifying *ALYA_PROCS* variable (you could try with 14, 28 and 56) and the cores per dislib task modifying *ComputingUnits* variable (you could try with 4 or 8).

Question 4: What is the impact of changing the number of Alya processes in the execution time? Why?

Question 5: What is the impact of changing the ComputingUnits in the execution time? Why?

Optional

Running with more samples

You can also try to execute a larger execution with 32 samples (*test_32samples.yaml*) and 4 or 8 computing nodes to see more real results. Be aware that for each execution it will take 10-20 minutes and we are sharing a reservation with all the students. Submit each test at a time to have a fair share of resources between the students.

Generating execution tracefile

You can also generate a tracefile to see more details of the execution. The execution trace provides a timeline representation of what are doing the different COMPSs processes and threads during the application execution. The run_tracing.sh script executes the submission command including the tracing flags. Run the same workflow execution to generate the execution trace

```
$ CAELESTIS > sh run_tracing.sh test.yaml 2
```

Once the execution is finished, load and run Paraver to visualise the execution trace.

WARNING: Be aware that the remote visualisation of trace files can be slow. You can install Paraver in your laptop downloading it from here. You will need to copy the trace files from *.COMPSs/<job_id>/trace/* and configuration files from *cfgs/*.

```
$ CAELESTIS > module load paraver
$ CAELESTIS > wxparaver .COMPSs/<job_id>/trace/*.prv
```

To analyse the trace, you can load configuration files by selecting *File -> Load Configuration* and then selecting one of the following configuration files:

- **compss_tasks.cfg**: it displays a timeline, where each line corresponds to a process in the execution and each different task type is displayed with different colours. To see the legend, select with the right click the *Info panel* option in the menu that is opened. Then, select the *Colours* tab. The MPI tasks are displayed as red with the name *alya_simulation*.
- **nb_executing_tasks.cfg**: this configuration file displays a single line that accumulates the number of tasks that are executed at a time.
- **2dp_tasks.cfg**: this configuration file displays a table with information about the tasks. You can see different statistics by opening the menus under the second icon on the right under the *Files & Windows Properties* area of Paraver.

Question 6: Trace analysis

a) How many MPI tasks (alya) are executed in parallel?	
b) What is the maximum number of parallel tasks?	
c) Which part is dominating the execution?	