Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

SHAPE Project Open Ocean: High Performance Processing Chain - faster on-line statistics calculation

L. Falletti[a*], S. Therond[a], Y. Moisan[b], Y. Kervella[b]

[a]*IDRIS – Orsay, France*
[b]*Open Ocean, France*

Abstract

Open Ocean is a French SME company which develops innovative on-line solutions to help plan and manage offshore developments. They conceived an oceanographic data study tool which computes and formats data (Pre-Processing and Processing) and which provides relevant oceanographic information to industrial marine companies (Post-Processing) through a web interface. But the "time-to-solution" of this post-processing step is too long and hence not compatible with industrial use. Therefore, the goal of this SHAPE project is to improve post-processing by optimising a parallelized Python program of Open Ocean which processes and computes statistics (e.g. wind speed) on big datasets. To carry this out, engineers of Open Ocean and IDRIS worked together to optimise this program by using resources available in a national supercomputing centre: high performance parallel machine and parallel file system (GPFS, 100 GB/S bandwidth).
This paper describes the parallelisation process implemented by Open Ocean and its porting on the Ada machine (IBM cluster of Intel E5-4650 processors, 332 compute nodes) at IDRIS. It also covers performance testing and identification of the bottleneck in the execution.

1. Introduction

The Earth's oceans cover 71% of its surface and are more than ever a key resource for energy (marine and fossil energy), food (fish and seaweed), transport and pharmaceutical (marine biology) sectors but remain a complex and hostile environment. Against this background, Open Ocean is a French company which has conceived innovative online solutions since 2011 which will change the way offshore developments are planned and managed [1].
This marine energy consultancy specialised in ocean numerical modelling for the marine energy and offshore wind sectors. Based on high resolution ocean numerical modelling, statistical analysis and data mining, Open Ocean provides a large catalogue of services both for the planning phase with the Metocean Analytics online offer (resource assessment, site characterization...) and the operational phase. These services are available for any location throughout the world.
Open Ocean has gained extensive knowledge of the challenges encountered by tidal project developers while providing key oceanic data and consultancy (for instance to SSER in the Orkneys or Futures Energies in the Raz Blanchard). Its goal is to advise clients on issues from potential market and resource assessment to tidal farm yield estimates. Open Ocean is involved in several research projects related to the marine energy industry focusing on the methodology of resource assessment, the impact of turbulence on tidal array design and multi-scale ocean numerical modelling. Furthermore, Open Ocean is a member of France Energies Marines and a 2012 recipient of the Young Innovative Company national contest organised by the French Ministry of Research.

Open Ocean uses a model where the ocean is discretised on a grid with hundreds of metres of spatial resolution. On each point of the grid, hydrodynamic equations are solved using ocean models developed in worldwide

[*] Corresponding author. *E-mail address*: lola.falletti@idris.fr

research centres. It can take several months to simulate 20 years of data on Open Ocean cluster (2 nodes) but once the simulations are performed, the bloc of data is analysed by the Open Ocean statistical toolbox.

In order to provide an on-line, on-demand tool, the whole Open Ocean Processing chain (pre-processing, processing and post-processing) has been automatized. The main problem to be tackled is the post-processing step. Indeed, at this time, it takes between 2 and 3 hours to extract time series data spanning 20 years and to analyse it ("statistics calculation") at 10 points of interest. As clients can require an analysis of several hundred points in the case of an offshore wind farm for instance, fast data access is vital for Open Ocean.
The post-processing step consists of a Python 3 program. The results produced by this program are then accessible from the Metocean Analytics [2]on-line offer through a user-friendly web interface (see Figure 1). This tool gives on-demand access to metocean[†] data, statistics and reports through essential analytics and displaying tools. Its key feature is that it permits a work-time gain of several weeks.

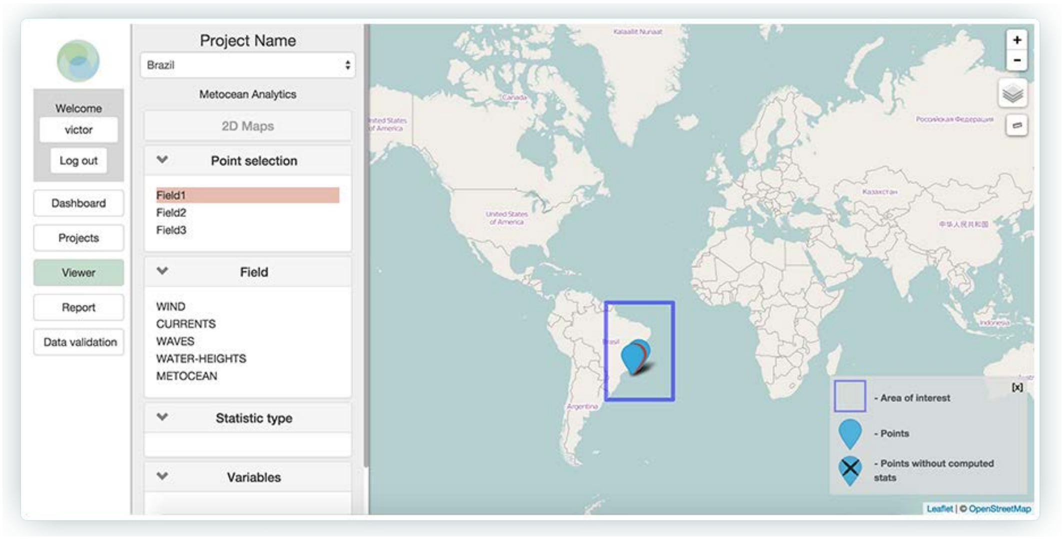

Figure 1: User-friendly Metocean Analytics web interface

The computing time of the post-processing step could be long (several hours) if the client requires an analysis of hundreds of points in order to know the metocean statistics for an offshore wind farm for instance. This computing time is too long for a web usage and is not compatible with an industrial use. To improve this tool, a profiling of the post-processing step is needed along with better parallelisation of data extraction and statistics calculations to identify the major bottlenecks. The aim of this study was to increase the speed of the whole process of extracting and processing data from datasets by parallelizing the execution of Open Ocean post-processing tools.

In the next sections, we will describe the steps of the work that was done in collaboration between Open Ocean and the IDRIS HPC (High Performance Computing) centre to enhance post-processing calculations.

2. Code porting at IDRIS

2.1. Description of the machine used

The Institute for Development and Resources in Intensive Scientific Computing (IDRIS), founded in 1993, is a service-based structure assuring the implementation and operation of a high performance, calculation-intensive environment designed to meet the great scientific challenges of numerical simulation. The machine group operated by IDRIS now consists of two architecturally complementary supercomputers ("Turing" and "Ada") owned by GENCI ("Grand Equipement National de Calcul Intensif") and were installed at the end of 2012 [3]. Of the two, Ada (see Figure 2) is the most appropriate machine for the working with Open Ocean as it has the

[†] Metocean is the abbreviation of the two words "Meteorology" and "Oceanography".

most wide-ranging usage. It is composed of large memory SMP nodes (IBM x3750-M4) interconnected by a high-speed InfiniBand network. Each compute node contains four 8-core Intel Sandy Bridge E5-4650 processors (32 cores per node) with a clock speed of 2.7 GHz and 2 connections to the InfiniBand FDR10 Mellanox network. Finally, the General Parallel File System (GPFS), a high-performance clustered file system developed by IBM, is used.

Figure 2 : The Ada machine (© CNRS photo library/C. Frésillon)

2.2. Simple test of the post-processing chain (sequential)

In order to operate the post-processing step, some Python 3 libraries had to be installed on Ada, such as PyQt4, GDAL, lxml, wafo and others. The size of the dataset downloaded for a simple but realistic case is almost 1.4 TB. The dataset contains the physical data (netCDF4 files) on which the statistical calculations will be done: wind, wave and flow data for the scenario treated in this study.

The first compute test (hereafter named "Test 1") consists of extracting 7 types of statistics at 2 points of interest. What is called "statistic" corresponds here to what will be treated by the software (extraction, time series computing, visualisation, exportation…) and has no link with mathematics. For example, the statistics for this first test are: Distribution Rose, Empirical Probability Description, Empirical Joint Probability Distribution, Extreme values, Times Series and Weather Window.
All the details which are needed to extract and analyze the data are contained in an XML file, hereafter named XML_config_file. This parameter file, also called "statistics file", specifies the following information:

- Datasets and associated variable on which to work
- Points of interest (longitude and latitude)
- Observation period
- Statistics which will be calculated or executed.

The dataset and the parameters file are used by the Python script Projet.py to export computation results (mainly generated in netCDF4 format) and visualisations (graph or table representing treated data). The schema below summarises this chain calculation (Figure 3).

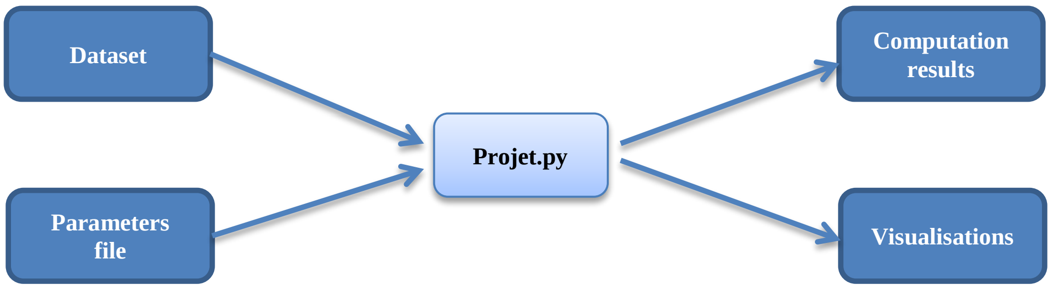

Figure 3: Input and output files of Projet.py

A report (55 pages for this first case) is then generated by the Metocean Analytics web tool with details and explanations about every statistic, result and graphic.

For the code porting on Ada machine, it was necessary to have a specific configuration file specifying the path to the dataset. Indeed, this situation was foreseen: An option was created in order to include this second configuration file. The Python script Projet.py is then used like this:

```
python3  Projet.py  <FOLDER>  <XML_config_file>  --runs_xml  <XML_spec_config_file>
```

The option <FOLDER> corresponds to the path of the folder where the calculation will be done; <XML_config_file> is the statistics file and <XML_spec_config_file> is the specific file for the Ada machine.

For the "Test 1" case (computation of 7 statistics at 2 points of interest), the calculation takes **61 minutes** on Open Ocean's machine against **51 minutes** on Ada machine. The difference of calculation time may come from the difference between the file systems (NFS versus GPFS) and/or the processor type.

To improve the compute time, we will now focus on the parallelization of the tasks.

3. Post-processing parallelization description

The script Projet.py is already partially optimised. Indeed, before the step of a statistic computation (the time series for instance), the script checks if the data needed by the statistic are already extracted (as "pickle" format). But this method remains sequential: The statistics will be computed one after the other. To parallelize the calculation, the idea is to have one job for each statistic, as many jobs as data extractions and to execute these jobs in parallel whenever possible. That is, an execution plan will be generated to identify the data that are used and pre-processed by several statistics to then arrange the jobs executions. Until now, the submission and execution of these jobs were run by the software ProActive Parallel Suite [4]. This software does include high-performance workflows and application parallelization but, unfortunately, it is hard to install without help from the company commercial support.

To summarize, two steps are needed to parallelize the post-processing computation:

- Creation of a dependency tree which will be export to a XML file format.
- Job submission and execution.

Using the ability of their local scheduler to wait for the end of a process before launching a new one, Open Ocean implemented a script which generates a workflow plan (a tree of dependencies) for each project. When executed, this plan forces some tasks to wait for others to finish in order to use previously processed data. The main algorithm of this script is simple:

- If several tasks use the same processed data, pick one that will be launched before all the others. These "commonly used data" must be serialized into pickle format.
- All other tasks which use these data will start after the first one finishes: When launched, they must first try to load existing serialized pickles instead of directly processing those data.

This parallelization method relies on the ability to predict the need of each task (in term of "commonly used data"). Indeed, this is possible thanks to the main parameter file XML_config_file (see previous section) which gives a summary of all the statistics (i.e. tasks) needed for a project, and each variable needed to compute these tasks. By parsing this file and by using the previously described algorithm, Open Ocean implemented the make_workflow.py Python script, which generates an execution plan as an XML file, understandable by the scheduler (workflow.xml):

```
python3  parallelization/make_workflow.py  <XML_config_file>  -o  workflow.xml
```

Figure 4 shows the tree of dependencies generated by make_workflow.py script for a simple project. In this diagram, each dependency is represented as an arrow. The top task *SYMLINK_CREATOR* is executed before all others: The aim of this task is to create the directory structure into which each task will export its output files (in order to avoid concurrent directory creations). When it has finished, *WIN_mag10_380*, *CUR_theta_TS* and *WAV_fp_TS_380* are launched simultaneously, and so on.

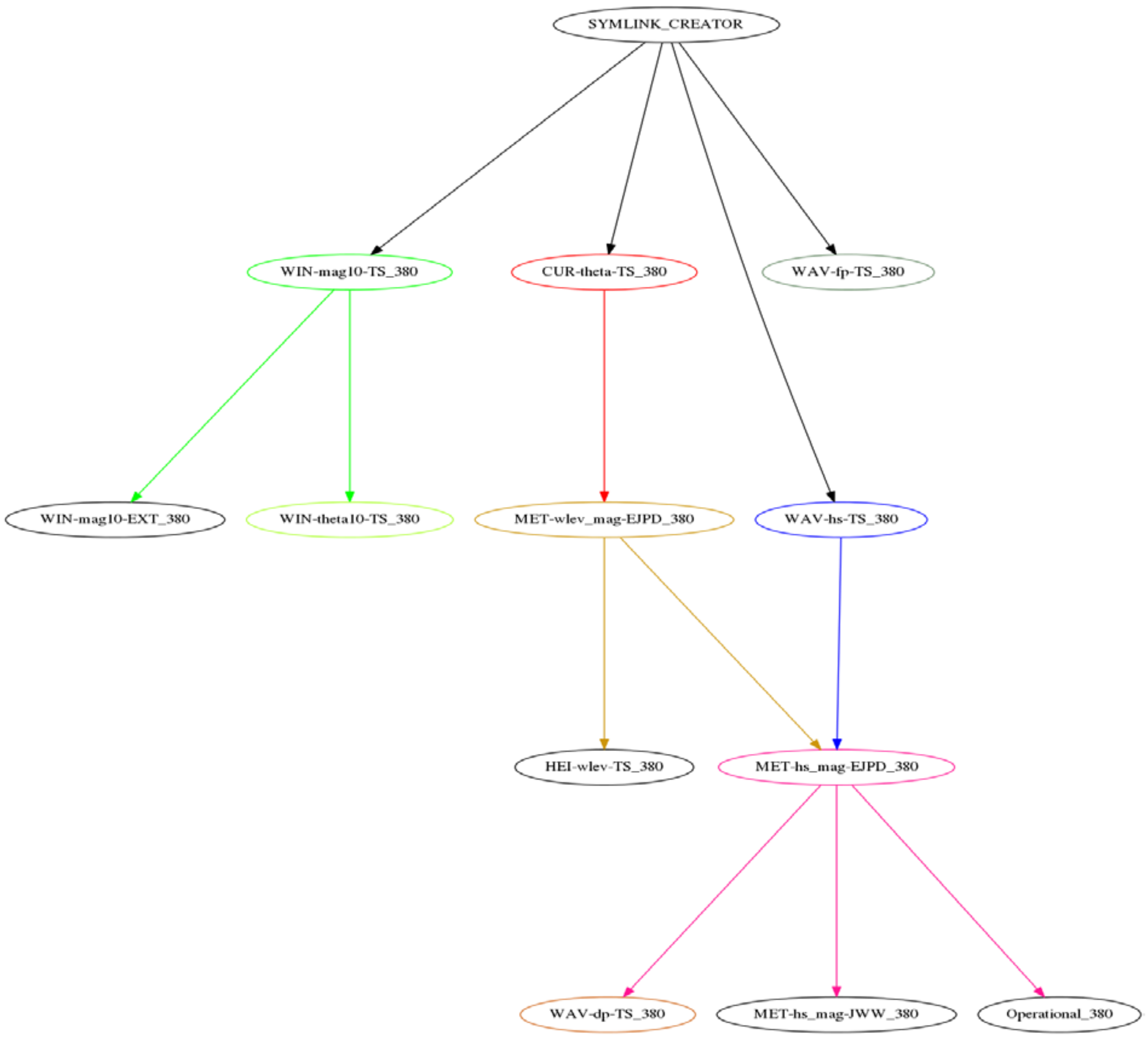

Figure 4: Visual representation of a dependency tree exported by 'make_workflow.py'.

The output file workflow.xml is an even simpler example and is shown in Figure 5. In this example, the task *WIN_mag10_380* corresponds to the wind statistics (wind speed and direction) and is dependent on the task *SYMLINK_CREATOR*.

For the “Test 1” case, a dependency tree was generated by the Python script previously described and was used by the scheduler installed on the Open Ocean machine. The calculation time is **30 minutes** on this machine (instead of 61 minutes in sequential mode).

```
<job options=…>
<taskFlow>
  <task name="WIN_mag10_380" runAsMe="true">
    <depends>
      <task ref="SYMLINK_CREATOR"></task>
    </depends>
    <nativeExecutable>
      <staticCommand value="bash">
        <arguments>
          <argument value="./Step/WIN/command"></argument>
        </arguments>
      </staticCommand>
    </nativeExecutable>
  </task>
  <task name="SYMLINK_CREATOR" runAsMe="true">
    <nativeExecutable>
      <staticCommand value="bash">
        <arguments>
          <argument value="./Step/SYMLINK/CREATOR/command"></argument>
        </arguments>
      </staticCommand>
    </nativeExecutable>
  </task>
</taskFlow>
</job>
```

Figure 5: Example of a simple tree of dependencies using XML format

4. Activity done during the SHAPE project: post-processing optimisation

4.1. Code porting on the Ada machine at IDRIS

To enable porting the post-processing code on Ada (and by extension to other machines in the PRACE program), it was necessary to adapt the software to the machine in order to avoid the challenge of installing ProActive scheduler on Ada without using the company for-pay support. It was decided to use the XML file generated by the make_workflow.py Python script. Indeed, this file represents the tree of dependencies: All of the dependency tasks are listed and thus can be used. The IDRIS team had developed a Python script export_xml_to_job.py to convert this XML file to a file readable by a machine job scheduler: LoadLever[‡] for Ada and SLURM[§] for other machines. It is possible for both LoadLever and SLURM to monitor multi-step jobs, i.e. to submit a job only if a previous job (or several) has been finished. The usage of export_xml_to_job.py is described below (Figure 6).

```
>>> python3 export_xml_to_job.py --help
usage: export_xml_to_job.py [-h] -i inputXMLFileName (--loadleveler | --slurm)

Create a multistep LoadLeveler or SLURM job from a workflow XML file

Optional arguments:
  -h, --help            Show this help message and exit
  -i inputXMLFileName, --input inputXMLFileName       Workflow XML file (required)
  --loadleveler         Create a LoadLeveler job file of the workflow.
  --slurm               Create a SLURM job file of the workflow.
```

Figure 6: Usage of the Python script export_xml_to_job.py

[‡] LoadLeveler is the job scheduler specific to IBM machines.
[§] SLURM is an open-source job scheduler designed for Linux clusters of all sizes.

For dependent jobs, the LoadLeveler scheduler gathers multiple job steps together in a single queue script: Each step is defined in a sub-job to which are associated its own resources (cores, memory, calculation time). The dependency of one job to another job (or several) can be specified using the directive "#@dependency". The writing of the file must obey the following schema: First, all the steps are defined with explicit directives for the LoadLeveler workload scheduler (resources, and dependencies if needed); then the batch commands to be executed in the different steps (sub-jobs) will be specified within a "case" bash instruction. A very simple LoadLeveler script generated by export_xml_to_job.py is shown below (Figure 7). In this example, the sub-job "Step 2" is dependent on Step 1: This sub-job will only be submitted once the sub-job "Step 1" is finished.

```
#========== Global directives ==========
# @ job_name = OPENOCEAN

#========== Step 1 directives ==========
# @ step_name = SYMLINK_CREATOR
# @ job_type = serial
# @ wall_clock_limit = 01:00:00
# @ queue

#========== Step 2 directives ==========
# @ step_name = WIN_mag10_380
# @ dependency = (SYMLINK_CREATOR == 0 )
# @ job_type = serial
# @ wall_clock_limit = 01:00:00
# @ queue

case ${LOADL_STEP_NAME} in
    #=========== Step 1 commands ===========
    SYMLINK_CREATOR)
     ./Step/SYMLINK/CREATOR/command
    ;;
    #=========== Step 2 commands ===========
    WIN_mag10_380)
      ./Step/WIN/command
    ;;
 esac
```

Figure 7: Example of the transcription of a simple dependency tree for LoadLeveler job scheduler

Contrary to Loadleveler, the SLURM job scheduler requires a single script per sub-job: The dependencies are specified during the job submission. If we follow the same simple example, the script generated for SLURM will be:

```
>>> cat SYMLINK_CREATOR.bash
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH -J SYMLINK_CREATOR
#SBATCH --time=01:00:00

./Step/SYMLINK/CREATOR/command
```

```
>>> cat WIN_mag10_380.bash
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH -J WIN_mag10_380
#SBATCH --time=01:00:00

./Step/WIN/command
```

Figure 8: Example of the transcription of a simple dependency tree for SLURM job scheduler

A final file (Figure 9) is then generated to facilitate the submission of each SLURM job: The execution of this single file will automatically submit the SLURM jobs with the correct dependencies.

```
>>>cat job-for-slurm.bash
#!/bin/bash
JID_SYMLINK_CREATOR=`sbatch  SYMLINK_CREATOR.bash | cut -d " " -f 4`
JID_ WIN_mag10_380=`sbatch  --dependency=afterok:$JID_SYMLINK_CREATOR WIN_mag10_380.bash
                                        | cut -d " " -f 4`
```

Figure 9: Example of the submission of dependent jobs using the SLURM job scheduler

Using this script, the submission of multi-step jobs is now possible on any machine which has LoadLeveler or SLURM installed. Furthermore, it can be easily adapted to other machine job schedulers. The calculation time of the "Test 1" case is now only **12 minutes** on the Ada machine at IDRIS.

To go further on testing the performances, a bigger case was used. This "Test 2" case computes the same statistics as for Test 1 (7 statistics) but at 100 points of interest. The results of the Test 1 and Test 2 computations are shown in Table 1. The calculation using the IDRIS machine is 2.5 times faster for Test 1 than using the Open Ocean machine, and 2.7 times faster for Test 2.
It is important to highlight that for the second case, a dedicated node was used for the computation on the Ada machine to have an exact performance comparison with the Open Ocean machine: 32 CPU are then used in parallel on both machines.

Job submission type	**Test 1 duration (minutes)**	**Test 2 duration (hours)**
Sequential Open Ocean	61	x
Sequential IDRIS (Ada)	51	x
Multi-step Open Ocean	30	4h40
Multi-step IDRIS (Ada)	12	1h44

Table 1: Summary of calculation results for two test cases. "Sequential" refers to what was explained in paragraph 2.2 and "Multi-step" in paragraph 4 of this document. Test 1 consists of the computation of 7 statistics at 2 points of interest. Test 2 consists of computation of the same 7 statistics at 100 points of interest.

4.2. Post-processing optimisation

To try to better understand the computation behaviour, we will focus on the Test 2 "multi-step" case launched at the IDRIS computer centre. For this case, 3700 steps were needed to compute the post-processing data. Figure 10 shows the histogram of the number of jobs in function of their computation times.

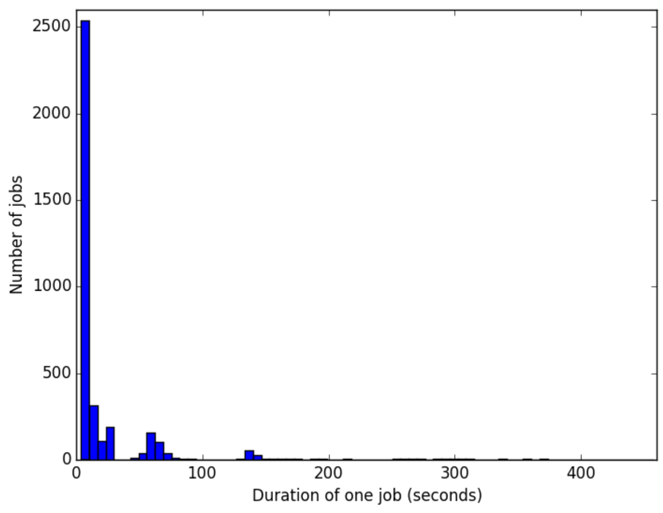

Figure 10: Histogram of the number of jobs in function of their computation times on the Ada machine for Test 2 case.

One can see that most of the jobs last less than 30 seconds. The longest computation is about 7min40s, and no more than about 20 jobs last more than 5 minutes, whereas 2500 jobs last 12 seconds or less. With this information and other details about what kind of task need more or less time, the next step would be to test how to optimise the computation itself: One solution for Open Ocean could be to use more than 32 CPU for the post-processing computation; another could be to re-organise and to optimise the computation script.

5. Conclusions

Benefits of PRACE cooperation:

The PRACE cooperation gave Open Ocean the opportunity to port their codes into a high performance computer system, thus confronting them with the standards of this computer science field. The in-depth knowledge of IDRIS engineers also gave Open Ocean a new look at both their hardware and file transfer solution.

Benefits for Open Ocean SME:

This study allowed Open Ocean to identify the main bottleneck of its post-processing program (i.e. fetching data from their dataset) and to reconsider their hardware choice. Also, by porting the post-processing code to the IDRIS infrastructure, this PRACE project gave the opportunity to Open Ocean to try and assess other job schedulers such as SLURM or LoadLeveler, which highly increases the portability and the efficiency of their software solution.

References

[1] http://www.openocean.fr/en/
[2] http://www.metocean-analytics.com/
[3] http://www.idris.fr/eng/
[4] http://proactive.activeeon.com/

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EU’s Horizon 2020 research and innovation programme (2014-2020) under grant agreement 653838.