IS-ENES3 Deliverable D9.3

ESMValTool version with ESGF coupling and distributed computation features

Reporting period: 01/07/2020 – 31/12/2021

Authors: Valeriu Predoi (UREAD), Bouwe Andela (NLeSC), Niels Drost (NLeSC), Rémi Kazeroni (DLR), Javier Vegas-Regidor (BSC), Carsten Ehbrecht (DKRZ), Stephan Kindermann (DKRZ)
Reviewers: Kim Serradell (BSC), Ag Stephens (UKRI CEDA)
Release date: 09/12/2021

ABSTRACT

This report covers the development, functional testing, and deployment of ESMValTool on compute nodes of HPC clusters that also host ESGF data nodes. These developments include the capability to run ESMValTool using data from local ESGF nodes and to download missing data from remote ESGF nodes. Distributing ESMValTool recipe's tasks across compute nodes linked to ESFG nodes has been enabled to preprocess data locally before download. The deployment of ESMValTool includes among others pre-installed modules, containerization and Jupyter notebooks to enhance user-friendliness and account for different infrastructures on HPC clusters. The enhancements to ESMValTool also comprise the generation of a webpage to display the results, including their provenance, through web servers. A new release of ESMValTool containing these developments has been made available (version 2.4.0). We will strive to integrate functionality for cross-HPC cluster distributed run capability in the upcoming stable release version 2.5.0.

Revision table			
Version	**Date**	**Name**	**Comments**
Release for review	24/11/2021	Valeriu Predoi	
Responses to internal review	09/12/2021	Valeriu Predoi	
Dissemination Level			
PU	Public		X
CO	Confidential, only for the partners of the IS-ENES3 project		

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084

Table of contents

Executive Summary

This report covers the development, functional testing, and deployment of ESMValTool on compute nodes of HPC clusters that also host ESGF data nodes. These developments include the capability to run ESMValTool using data from local ESGF nodes and to download missing data from remote ESGF nodes. Distributing ESMValTool recipe's tasks across compute nodes linked to ESFG nodes has been enabled to preprocess data locally before download. The deployment of ESMValTool includes, among others, pre-installed modules, containerization and Jupyter notebooks to enhance user-friendliness and account for different infrastructures on HPC clusters. The enhancements to ESMValTool also comprise the generation of a web page to display the results, including their provenance, through web servers. A new release of ESMValTool containing these developments has been made available (version 2.4.0). We will strive to integrate functionality for cross-HPC cluster distributed run capability in the upcoming stable release version 2.5.0.

The ESMValTool's capability to use local ESGF nodes allows for minimal data movement, whereas the option and capability to download and cache data from remote ESGF nodes allows for maximum data coverage of diagnostic runs at any one location, giving the user full access to perform their diagnostics runs on an optimized HPC cluster. Data availability as a whole complements ESMValTool's capability to run parallel preprocessing tasks, thus minimizing its computational resources footprint; this capability is also described in this report.

Improved user-friendliness is guaranteed by providing several solutions to significantly reduce installation efforts for ESMValTool users and allow the use of the software on multiple diverse platforms and interfaces, including modern Jupyter notebooks. We also describe some aspects related to the visualization of results produced by ESMValTool through its capability to generate a web page to browse recipe results and view the documentation of the recipe, the produced figures and download provenance data.

1. Objectives

The ESMValTool aims at supporting very diverse use cases, ranging from running the Tool on HPC clusters with locally available data, to distributing computations across HPC clusters based on data availability, and to running the Tool and visualizing results through web services. The capability of reading data stored on an ESGF node attached to a given HPC cluster and accessing missing data stored on other ESGF nodes is of paramount importance for the ESMValTool. The objective of this work is to tightly couple ESMValTool to HPC centers linked to ESGF nodes so that data availability is no longer an issue for the user while minimizing the computational resources footprint and data movement.

In the following sections, we describe how ESMValTool has been coupled to ESGF nodes by covering several case studies and deployment efforts done at some major HPC sites. First, we focus on how ESMValTool has been enabled to use ESGF data available on a given cluster (section 2.1) before reporting on the implemented capability to download missing data from other ESGF servers (section 2.2). We also describe several strategies used to deploy ESMValTool on HPC clusters to enhance user-friendliness and reduce installation efforts for the users (section 2.3). Furthermore, we report on the added capability to distribute ESMValTool's tasks across computing clusters in order to preprocess ESGF data locally before download (section 2.4). Finally, we focus on different use cases of ESMValTool at DKRZ which provide nearly complete access to ESGF data (section 2.5) and also describe the possibility to display results and their provenance through a web interface (section 2.6). In section 3, we draw overall conclusions and make recommendations for next steps.

2. Coupling ESMValTool to ESGF data nodes

2.1 Case study: ESMValTool running in the close vicinity of an ESGF node

This case study proposes the following scenario: given an HPC cluster, that has an ESGF data node attached to it (accessible both via the file system, and via a web interface), we would want ESMValTool to run centrally on a user-accessible disk partition, on said HPC, with the Tool using data from the attached ESGF node as input to the various user-invoked diagnostic runs. This section fully describes the current implementation of this used case. In order for ESMValTool to perform this task, one needs the following functional specifications:

- the Tool needs a data finding module that uses a configuration specific to ESGF data storing conventions, finds the user-defined data on the ESGF node, retrieves it, and passes it to the data preprocessing modules that are run before the actual diagnostics (ESMValCore);
- this module should be easily configurable to account for variations in ESGF data storing conventions (these conventions differ slightly from node to node); it should also be flexible enough to use custom-made data storing conventions (e.g. in case of locally-stored CMIP

data, or centrally-stored observational data that follows completely different storage conventions);

- this module should require minimal input from the user *i.e.* it should function with a minimal set of parameters describing the data, and perform the rest of the work finding and retrieving data in an automated, scalable and portable (across ESGF nodes) manner;
- being part of ESMValCore, and being a vital module for the Tool's functionality, this module should be thoroughly tested via automated testing modules, with testing coverage aimed at 100%, and each new change-fix implementation should come with a number of tests to cover it fully.

We have implemented such a module and it has been part of the ESMValCore's (and consequently, part of ESMValTool) workflow since the release of version v2.0 [1]; documentation for its functionality and on how the user should configure it can be found in the data finding section of the documentation. To note that the data finder module was one of the very first modules that UREAD has implemented part of the original work to start developing ESMValTool v2.0, all the way back to mid-2017, but its original form has changed quite heavily since its first iteration: we have optimized it for speed, we have also generalized it so that data with different storing conventions can be found, even more than that – now, a user can input wildcard elements to retrieve not just a single file but rather, a subset of files covering multiple variations of data parameters; we have also optimized the data search algorithm to account for very specific issues related to CMIP6 data (e.g. retrieving data for `fx`-type variables (auxiliary variables that describe masks, cell areas and cell volumes); these variables suffer from a number of issues with the way they are stored on ESGF data nodes, and don't usually adhere to the CMIP6 data storing standards). We have also written a comprehensive number of automated tests for this module, that can be triggered by the user or developer whenever they add or fix a functionality in the module, and also run nightly, part of our Continuous Integration (CI) pipelines, deployed on Circle (see the Circle CI page, needs login with GitHub credentials) and GitHub Actions (see Github Actions Tests) platforms, currently with an estimated test coverage of 95% (see test coverage break-down in the Codecov Coverage Report).

Regarding observational data needed for an ESMValTool run, we strive to maintain and curate pools of such datasets on the HPC clusters CEDA-JASMIN and DKRZ/Mistral close to ESGF nodes. Every newly supported dataset by the ESMValTool is added to the shared groupspace of ESMValTool developers on DKRZ/Mistral and this is then mirrored to CEDA-JASMIN. The pools of data are regularly curated to account for datasets no longer available on source servers or newer versions supported by the ESMValTool. We are currently working on an extension of the observational data handler in ESMValTool, meant for users that don't have access to these HPC clusters, that will allow the local download of OBS data requested by the user.

2.2 Case study: ESMValTool running in the close vicinity of an ESGF node but some data is missing from the node: implementing `esgf-pyclient`

Running the tool on HPC clusters like CEDA-JASMIN or DKRZ/Mistral is beneficial to the user in that it allows access to very large amounts of CMIP data (and, depending on the cluster, considerable amounts of Observational datasets), locally stored, on an ESGF data node available on connected File Systems, and accessible via command line interface, through a shared OS, as we have pointed out in the previous section.

However, in terms of the evolution of the data finder/ESGF-coupler module, we have identified an immediate enhancement that is conditioned by the lack of full coverage of CMIP data on any given ESGF node: an average 90% of published CMIP data coverage is typical of ESGF nodes but the question arises – what to do when the user needs data that is not present on the node that ESMValTool has access to via a shared file system? The missing odd 10% may amount to hundreds of Gigabytes of missing data. One of the main purposes of ESMValTool is to provide fully reproducible results, no matter the site where it is run at, so it is imperative for the user that needs to reproduce a set of results to have access to the entire data specified in the recipe they need to run; if 10% of the data in that recipe is missing, the results are rendered impossible to reproduce, or, at best, partially reproducible. For this purpose we have implemented an automatic data download functionality in ESMValTool, functionality that uses the ESGF Pyclient tool as backend (see ESGF-PyClient documentation and consult their GitHub repository); the tool is installed as a dependency of ESMValCore (currently in `noarch` architecture *ie* common to Linux, Windows, and OSX architectures) from PyPi, and provides an interface and API allowing the download of data from various remote ESGF nodes via secure shell connections, providing API for interactions with the ESGF Search API). Configuration of this backend from the user perspective is minimal (see configuration in ESMValCore), and the search/download process benefits from a high degree of automation. Specifically, for a minimal working configuration, the users need only to define a handful of runtime variables ie either a flag to the run command `--offline=False` and a download directory name in the configuration file or `offline: false` and the download directory e.g. `download_dir: ~/climate_data` in the configuration file (`config-user.yml`). We note that the automated download functionality is turned off by default, and needs to be specifically called by the user; this eliminates unnecessary data transfers and the accumulation of data in repositories that are accessible only by the user (e.g. `$HOME`) in cases when either wrongly defined paths to available, centrally-stored data are used or the user is not providing ESMValTool with paths to data they have already downloaded (e.g. they have renamed such directories, and are now outside ESMValTool runtime environment). We have started establishing a liaison mechanism between users requiring missing data (and eventually downloading it with ESMValTool) and HPC staff responsible for data replication, just so that we avoid such cases. We also note that the automated download benefits from a few performance refinements: the two most useful for the user are: downloading data from

the "closest" ESGF node, based on measuring the shortest server response times, and ordering the servers based on this parameter, and the automatic detection of data that had already been downloaded, so that next time ESMValTool runs, that data will just be read from the local disk. Data is stored in standardized ESGF DRS paths that are replicated on local disk as they are found on the remote servers; this directory structure may be somewhat different than the standardized DRS at the local ESGF node where ESMValTool is running e.g. the tool is running on CEDA-JASMIN, for which the local DRS is set by the user in their configuration file to `BADC`, but some data was downloaded from DKRZ's ESGF node, so it comes packed with a different directory structure than the data on CEDA-JASMIN (`DKRZ`, specific DKRZ DRS), therefore the user should set the unified DRS to be `ESGF`, in their configuration file, to allow the use of both directory structures. Assuming the user chooses the downloaded data to reside in a `~/climate_data` master directory, then the `rootpath` and `drs` configuration parameters can be set as shown in Figure 1. More details about DRS paths and ESGF can be found in the ESMValCore documentation.

```
# Directory tree created by automatically downloading from ESGF
#rootpath:
  CMIP3: ~/climate_data
  CMIP5: ~/climate_data
  CMIP6: ~/climate_data
  CORDEX: ~/climate_data
  obs4MIPs: ~/climate_data
drs:
  CMIP3: ESGF
  CMIP5: ESGF
  CMIP6: ESGF
  CORDEX: ESGF
  obs4MIPs: ESGF
```

Figure 1: Screenshot showing configuration in `config-user.yml` that allows the setting of `rootpath` and `drs` parameters for data downloaded from remote ESGF nodes, as well as setting common data directories for CMIP, CORDEX, and obs4MIPs data to `climate_data`.

2.3 Deployment on HPC clusters and running parallel computations in the Preprocessor

The ESMValTool has been made available as a pre-installed package on the CEDA-JASMIN cluster and the Mistral cluster at DKRZ. To use the ESMValTool from the command line, users of these machines can load the existing module, by running the command `module load esmvaltool`. To discover the available versions of the `esmvaltool` module, use `module avail esmvaltool`. For example, on Jasmin this results in:

```
-------------------------- /apps/jasmin/modulefiles --------------------------
esmvaltool/2.4(default)
```

Support for older versions of ESMValTool is currently unavailable, since our aim is to provide users with the latest, stable version, in order to avoid backwards incompatibility issues. We will, however, strive to allow support for older versions, at least one or two releases older; this will allow the user to reproduce results obtained by peers that have run older versions in the past; the main module will still be the latest, stable version. Full install logs of all the current and previously installed versions of ESMValTool on CEDA-JASMIN are provided here.

A possible future extension of the central installation of ESMValTool would be central installations of tools like Synda (see Synda on GitHub and documentation). A central installation of Synda, callable by the user via (the already-existing, and tested) ESMValTool data download module with Synda will allow for central data replication, in a supervised, version-controlled manner (e.g. the `synda install` command); this will minimize data transfers and data duplication, and will serve as a direct link to the data replication team, that will be provided with an already existing replicates pool, with a correct ESGF DRS structure. As a first step, since ESMValTool already benefits from an integrated functionality using Synda for CMIP data downloads (as a somewhat less flexible alternative to the `esgf-pyclient` functionality, since the `install` Synda command is not available for user invocation), a possibility would be to use this existing functionality but with data replication in a central Synda directory. Synda developers are on the verge of releasing a refactored and much improved version of the tool, and we will aim to use that one when it becomes available.

In addition to using the ESMValTool from the command line, the ESMValTool can also be used from a modern Jupyter notebook interface at DKRZ. Instructions for accessing the DKRZ Jupyterhub can be found here and ESMValTool example notebooks (see Figure 2), to start off from, are available as an ESMValTool Jupyter example.

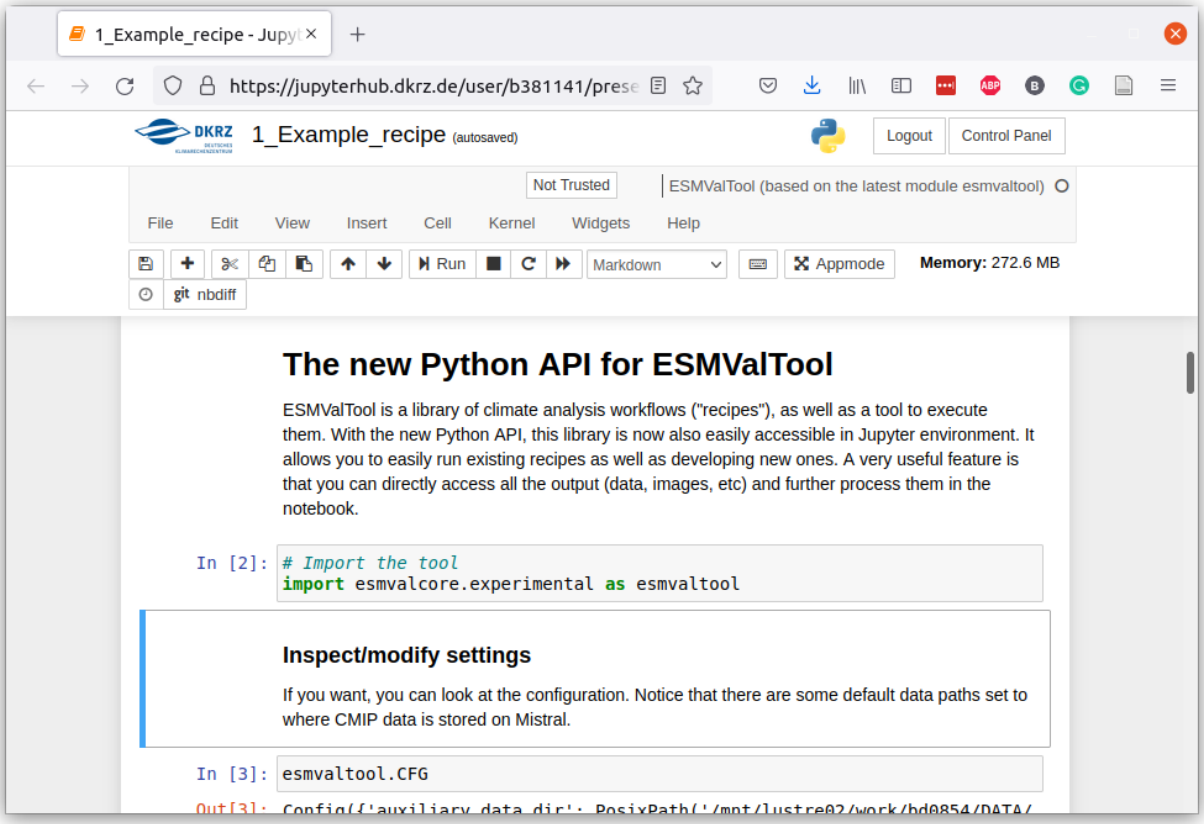

Figure 2: Screenshot showing the example Jupyter notebook running ESMValTool at DKRZ.

At BSC, on the other hand, container technology is used to deploy the ESMValTool. The administrator builds a Singularity container with the command:

```
singularity build /path/to/esmvaltool.sif docker://esmvalgroup/esmvaltool:stable
```

and then the users can run the tool from the `esmvaltool.sif` container with a command like

```
singularity run -B /data:/data /path/to/esmvaltool.sif run recipe_example.yml
```

where the `-B` flag is used to make files and directories from the file system available inside the container. More information is available in the installation instructions.

Finally, many active users and developers of the tool choose the installation from source method, with dependencies installed from conda-forge in a Python virtual environment. This allows for easy development since the active code can be modified in place, and the developer can just run and debug with the changes they've just made.

2.4 Distributed tasks across compute nodes linked to ESGF nodes at different HPC clusters

The ESMValCore module allows running a recipe in two stages, decoupling running the preprocessor from running the diagnostic script. This makes it possible to define preprocessors that reduce the input data volume, and run those on a machine that is close to the data. The resulting preprocessed data can then be downloaded and fed to the diagnostic scripts for the final analyses. This feature would also allow setting up a WPS (Web Processing Service) that just runs the preprocessing part of the recipe and returns the preprocessed data for the user to download to their local machine for further analysis. The use of this feature is described in the ESMValCore documentation on running the tool section.

To demonstrate this feature, the ESMValTool recipe shown below will be used. Some of the preprocessing is done on Mistral at DKRZ (the data from CMIP5 CanESM2 model) and some other data is preprocessed on CEDA-JASMIN (the data from CMIP6 BCC-ESM1 model). Finally, the preprocessed data is downloaded from both machines to a local machine and the plots are created there. Content of example distributed recipe `recipe_distributed.yml` used here:

```
documentation:
  title: Recipe that runs an example diagnostic written in Python.
  description: Example recipe that plots a map of temperature.
  authors:
    - andela_bouwe
  references:
    - acknow_project
  projects:
    - esmval
    - c3s-magic
    - isenes3

preprocessors:
  select_january:
    extract_month:
      month: 1

diagnostics:
  map:
    description: Global map of temperature in January 2000.
    themes:
      - phys
    realms:
      - atmos
```

```
    variables:
      tas_ceda:
        short_name: tas
        mip: Amon
        preprocessor: select_january
        start_year: 2000
        end_year: 2000
        additional_datasets:
          - {dataset: BCC-ESM1, project: CMIP6, exp: historical, ensemble: r1i1p1f1,
grid: gn}
      tas_dkrz:
        short_name: tas
        mip: Amon
        preprocessor: select_january
        start_year: 2000
        end_year: 2000
        additional_datasets:
          - {dataset: CanESM2, project: CMIP5, exp: historical, ensemble: r1i1p1}
    scripts:
      script1:
        script: examples/diagnostic.py
        quickplot:
          plot_type: pcolormesh
          cmap: Reds
```

To run the recipe above in a distributed way, the commands shown below were used.

- ran on Mistral at DKRZ:

```
esmvaltool run recipe_distributed.yml --diagnostics map/tas_dkrz --
remove_preproc_dir=False --run_diagnostic=False
```

This resulted in a number of log messages, with the following message indicating the output directory for the preprocessed data:

```
2021-11-18 16:22:41,906 UTC [2277] INFO       PREPROCDIR =
/pf/b/b381141/esmvaltool_output/recipe_distributed_20211118_162237/preproc
```

- ran on CEDA-JASMIN:

```
esmvaltool run recipe_distributed.yml --diagnostics map/tas_ceda --
remove_preproc_dir=False --run_diagnostic=False
```

This resulted in a number of log messages, with the following message indicating the output directory for the preprocessed data:

```
2021-11-18 16:18:27,104 UTC [27364] INFO      PREPROCDIR = /work/scratch-
pw/bandela/esmvaltool_output/recipe_distributed_20211118_161821/preproc
```

The resulting output directories `recipe_distributed_20211118_162237` and `recipe_distributed_20211118_161821` were then copied to a local directory `~/distributed`.

To run the actual analysis on the preprocessed data, the following command was used:

```
esmvaltool run recipe_distributed.yml --resume_from
"~/distributed/recipe_distributed_20211118_161821
~/distributed/recipe_distributed_20211118_162237"
```

This resulted in the two panels of Figure 3 for the average temperature in January 2000 according to CanESM2 and BCC-ESM1 models.

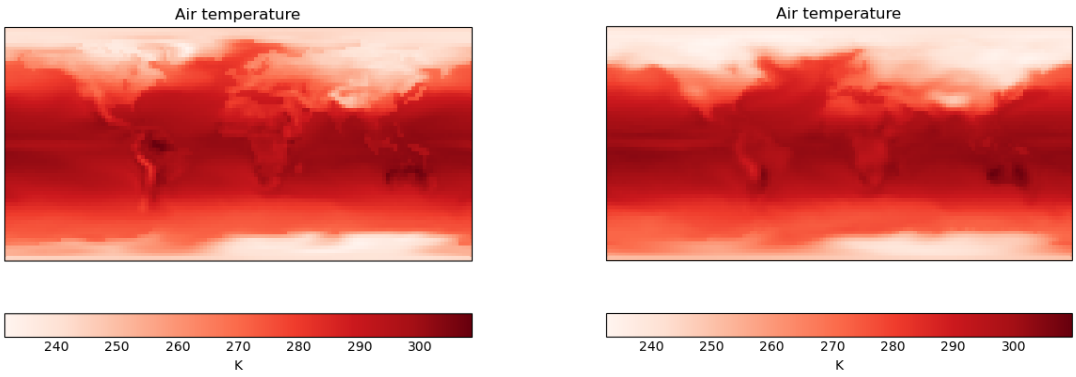

Figure 3: Average near-surface air temperature in January 2000 according to CanESM2 (left) and BCC-ESM1 (right).

and the following log messages were shown on the screen, which demonstrate that the preprocessed data was indeed used:

```
2021-11-18 16:25:09,843 UTC [32933] INFO      Re-using preprocessed files from
/home/bandela/distributed/recipe_distributed_20211118_161821/preproc/map/tas_ceda for
map/tas_ceda
2021-11-18 16:25:09,845 UTC [32933] INFO      Re-using preprocessed files from
/home/bandela/distributed/recipe_distributed_20211118_162237/preproc/map/tas_dkrz for
map/tas_dkrz
```

2.5 ESMValTool running near to DKRZ replica data pool

The latest released version of ESMValTool is available on the DKRZ HPC cluster Mistral. It can be used in two ways.

- It can be run on the command line by running the command `module load esmvaltool`.
- It can be used in a Jupyter notebook on the DKRZ Jupyter-hub

The usage details of both use cases are described in Section 1.3.

The ESMValTool has in both cases access to the complete data archive available on the Lustre filesystem at DKRZ. The local archive on disk includes more data than is available from the ESGF data node at DKRZ. Not all local replicas (CMIP6, CORDEX, ...) have been published on ESGF. It also makes it possible to replicate data demanded by the ESMValTool users on short notice without going through the ESGF publication process.

2.6 Visual displays of the evaluation results with provenance

Running an ESMValTool recipe results in the generation of a web page for easy viewing of the results. If the output directory is copied to a web server, the generated web page can directly be displayed and used for viewing the figures, as well as downloading the data used to create the figures, references, and provenance files. The results web page contains a description of the recipe, a list of authors, references, and, of course, the analysis results. Figure 4 shows the web page obtained after running `recipe_capacity_factor.yml`.

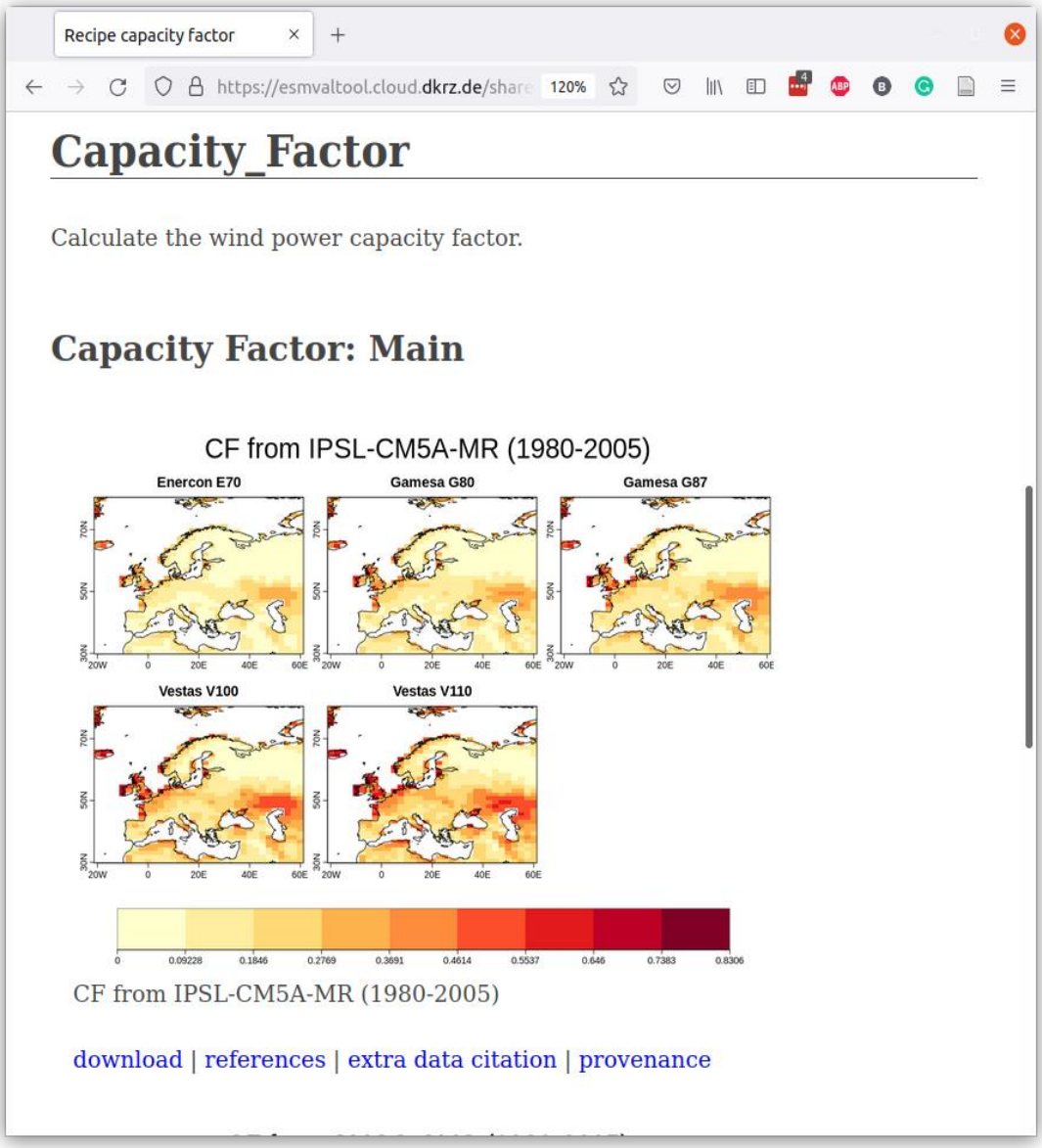

Figure 4: An example results web page, displaying results of `recipe_capacity_factor.yml`.

The four buttons shown in Figure 4 are links to download the files describing the figure, these are available for each output file: `download` - download the figure; `references` - a BibTeX file containing references to relevant papers describing methods and datasets used to create the figure; `extra data citation` - additional data citation information that was not available in BibTeX format; `provenance` - for each figure or file produced by the ESMValTool, provenance is collected and stored in W3C PROV-XML format. The provenance includes a list of input files used (and their global attributes), the preprocessing steps and settings that have been applied, references, authors, as well as various scientific labels (or "tags", as they are known in ESMValTool) describing the type of analysis.

3. Conclusions and Recommendations

In this report we have summarized a number of major developments of ESMValTool in conjunction with existing HPC data storage and software environments: we have shown the capability of the tool to run close to an ESGF data node, fully utilizing the CMIP data store on said node for diagnostic runs, without any need of third party data download applications or external data transfers; we have also described the integration of an existing Python-based ESGF client in the tool, making the automated download of locally-missing CMIP data from remote ESGF nodes a process available to and controlled by the ESMValTool user (with minimal configuration overhead, and optimized for transfer speeds); we have also shown that ESMValTool is now part of centrally-installed software ecosystems at major HPC sites like CEDA-JASMIN, and DKRZ, allowing the user to simply load an existing module and start running the tool; in the same vein, we have now centrally deployed a Jupyter notebook version of the tool, an interface that is becoming extremely popular with users. We have started the effort of implementing a fully (cross-HPC) distributed functionality for ESMValTool, to fully utilize the available computational resources, maximize data coverage, and minimize data transfers across HPC clusters, and, as a first step, we have implemented the re-use of preprocessed data generated by the Preprocessor, the computational engine of ESMValTool.

There are, however, a number of major improvements and new functionalities that we are targeting for the near future. We are listing here these improvements that are scheduled for upcoming ESMValTool releases, some of them being natural extensions of the current development process, others arising from the need for optimization and minimization of data transfers.

The automatic data download functionality, whereas an extremely useful tool at the user's fingertips, may produce a number of long-term issues related to multiple data transfers of the same datasets (by different users, each storing the data in locations that are not available to other users) and filling up disk space on clusters like CEDA-JASMIN or DKRZ. This is why we are developing a close bond with the data replication teams from each of these HPC sites: this will follow a feedback loop pattern - ESMValTool users provide us with details of their downloads, followed by us providing these details to the data replication teams, which, in turn, will make sure that data is available centrally on the cluster, and after that is done, we alert the users and ask them to remove any data they have downloaded, and further use the newly centrally-provided data. This interaction will be using a GitHub interface, with its dedicated project and the use of templated issues.

Observational (OBS) data is as important to the ESMValTool user as model data: currently we offer users access to centrally-hosted and curated OBS data repositories, located on CEDA-JASMIN and DKRZ (see obtaining observational datasets for ESMValTool). These repositories

contain large amounts of data (1.5 TB), however, they may be missing the OBS data that are needed by upcoming ESMValTool diagnostics, if the diagnostic requires the use of different datasets than those hosted centrally. ESMValTool benefits from an advanced functionality for converting raw OBS data to CMOR standards (minimal requirement for ESMValTool data processing), and extensive API to simplify this process for the user, but this process still relies on the user providing the actual raw data (this means downloading raw data from source, temporary data hosting on an HPC node while converting the data to CMOR standards, and removing it once the converted data has been ingested in the central OBS data repository). This process will benefit from a future inclusion of OBS data as part of the automatic download functionality; we already have such a functionality in its last stages of development and testing, further to be deployed part of the next release version 2.5.0: this allows for automatic raw OBS data download (provided the user inputs the source location), automatic conversion to CMOR standards, and automatic ingestion of the converted data in the ESMValTool diagnostic.

Distributing tasks across HPC clusters will benefit from the implementation of a user-controlled task manager, that will assign tasks according to required memory intake, run time requirements etc.; such a task manager, akin the current `cylc` facility already in use at various sites, but extended to managing inter-HPC tasks, will be developed in close collaboration with the cluster maintainers.

References

[1] Righi, M., Andela, B., Eyring, V., Lauer, A., Predoi, V., Schlund, M., Vegas-Regidor, J., Bock, L., Brötz, B., de Mora, L., Diblen, F., Dreyer, L., Drost, N., Earnshaw, P., Hassler, B., Koldunov, N., Little, B., Loosveldt Tomas, S., and Zimmermann, K.: Earth System Model Evaluation Tool (ESMValTool) v2.0 - technical overview, Geosci. Model Dev., 13, 1179-1199, https://doi.org/10.5194/gmd-13-1179-2020, 2020.

HPC Resources Websites

CEDA-JASMIN: https://www.ceda.ac.uk/services/jasmin/
DKRZ/Mistral: https://www.dkrz.de/up/systems/mistral

Abbreviations

API: Application Programming Interface
BSC: Barcelona Supercomputing Center
CEDA: Centre for Environmental Data Analysis

CMIP: Climate Model Intercomparison Project
CMOR: Climate Model Output Rewriter
DKRZ: Deutsches Klimarechenzentrum (German Climate Computing Center)
DLR: Deutsches Zentrum für Luft- und Raumfahrt (German Aerospace Center)
DRS: directory structure of the data
ESGF: Earth System Grid Federation
ESMValTool: Earth System Model eValuation Tool
HPC: High Performance Computing
NLeSC: Netherlands eScience Center
UREAD: University of Reading
W3C PROV-XML: World Wide Web Consortium Provenance data model in XML format