D6.6 Protocol for urgent HPC

Version 1.0

Documentation Information

Contract Number	9555558
Project Website	www.eFlows4HPC.eu
Contractual Deadline	29.02.2024
Dissemination Level	PU
Nature	R
Author	Miroslaw Kupczyk (PSNC), Steven J. Gibbons (NGI), Marisol Monterrubio (BSC), Louise Cordrie (INGV), Jacopo Selva (INGV), Carlos Sánchez (UMA), Cedric Bhihe (BSC), Josep de la Puente (BSC), Marta Pienkowska (ETH)
Contributors	Jorge Ejarque (BSC)
Reviewer	Rosa M Badia (BSC)
Keywords	Urgent Computing, crisis management, financial model

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

Change Log

Version	Description Change
V0.1	Draft, ToC
V0.2	Ready for the internal review
V0.3	Review comments addressed
V1.0	Document formatted for submission

Table of Contents

1 Executive Summary

This deliverable was originally planned to summarise the results of subtask "Access policy for urgent computing at HPC infrastructures". However, it has been identified that several other tasks regarding Pillar III implementation had no matching deliverable. Instead of proposing a new deliverable, it was the conclusion of the projects' EB that D6.6 would also include results from said tasks, thus giving closure to WP6 activities. Regarding the original urgent computing task, this deliverable presents the protocol for urgent access to Tier-0 systems. Automation rules have been considered, and an evaluation of the cost of different scenarios has been undertaken. Piloting recommendations for the governing bodies of such systems have been proposed in the external white paper. Availability and QoS of the computing ecosystem were discussed. The work is being carried out in order to help prepare an urgent computing contract between stakeholders.

Urgent computing is a specialised type of computing that prioritises and expedites the execution of time-critical tasks. Unlike traditional batch computing, where jobs are queued and executed in a sequential order, urgent computing provides immediate access to computational resources for critical applications that require timely analysis, decision-making, or response to real-world events. In the appendix section, there have been shown the examples of configuration files which are required to run the urgent applications being considered in the eFlows4HPC project.

Regarding Pillar III validation, we show specific cases of application of the PTF and UCIS4EQ workflows. We focus on aspects such as programmability, deployment and performance to exemplify how said workflows have benefitted from HPCWaaS developments within the project. The novel MLESmap workflow, unforeseen at the beginning of the project, is given similar treatment.

2 Validation of Use Case Tsunamis

The second and final iteration of the eFlows4HPC Probabilistic Tsunami Forecast (PTF) workflow was detailed in Deliverable 6.4. We here document the testing and validation of the workflow against the requirements detailed in Deliverable 6.1. The status of the requirements at the end of the project is provided in Table 1.

Table 1. PTF requirements analysis at the final stage of the project.

Name	Priority (must, should, may)	Current status	Comment on the current status
Urgent Computing Access	must	Addressed in this document.	Protocol for enabling urgent computing access.
Data accessibility	should	Implemented	help Automated with Yorc and DLS. Data sources are obtained from Data Catalog compliant with FAIR.

Data replication	must	Implemented	Within the limits of the semi-random scenario selection.
Execution robustness	must	Implemented	Tested and validated PyCOMPSs workflow.
Infrastructure interoperability	must	Implemented	Implemented through HPCWaaS (Alien4Cloud: detailed in D6.4)
Portability/Reusability	must	Implemented	Implemented through HPCWaaS (Alien4Cloud: detailed in D6.4)
Integrated Workflow Manager	must	Implemented	Tested and validated PyCOMPSs workflow.
Streaming Data Source	must	Incorporated in workflow design	No data listener implemented in the final eFlows4HPC PTF workflow because the TRL of processes that would update and interpret the external sensor data is currently too low. However, the design of the workflow is such that a data listener will be readily accommodated.
Integration with permanent storage	must	Implemented	Workflow outputs transferred to B2DROP.
Inference with online/offline ML	must	Implemented	Offline ML procedures demonstrated and documented (documented in D6.4).
DA Integration	may	Developed and tested but in isolation from the operational workflow.	An implementation with Ophidia that processes the workflow output has been written and tested but is not a part of the operational workflow due to incompatibilities between the Ophidia environment and that for the rest of the workflow.
Workflow malleability	should	Incorporated in workflow design	Progressive evaluation in place but functionality to generate new scenarios or cancelling scenarios not implemented.

2.1 Programmability of the Tsunami workflow

Prior to the eFlows4HPC project, the PTF workflow was implemented as a bash script (Appendix B) which invoked different MATLAB, python and bash scripts to process the different stages of the workflow (ensemble generation, configuration of input for the HySEA simulations, execution of the simulations in HPC resources, and post-processing the simulation output to compute the hazard curves). The driver code was rewritten in Python for seamless integration using PyCOMPSs. This solved a number of licence and portability issues, in particular with respect to the MATLAB code. The PyCOMPSs implementation is provided in Appendix C.

Those parts of the workflow that were already written in Python (e.g. the ensemble generation and post-processing) were implemented with great ease in the new workflow using the @task PYCOMPSs decorator. The bash script used to generate the input for Tsunami-HySEA was reused and invoked using the @binary PYCOMPSs decorator, and the tsunami simulations themselves were invoked using the @mpi PYCOMPSs decorator.

Table 2. Programmability metrics for the eFlows4HPC PTF workflow (using PyCOMPSs) and the original bash script-based workflow. LLoC stands for Logical Lines of Code (in which comments and split lines are ignored) and Cyclomatic Complexity is a complexity measure described by McCabe (1976).

Version	LLoC	Cyclomatic Complexity
PyCOMPSs	143	10
Original bash	225	34

Table 2 compares some programmability metrics for the PyCOMPSs workflow and the original bash-driven workflow. We have measured the Logical Lines of Code (LLoC), which counts the lines of effective code, discarding comments and detecting statements which are split in several lines, and the Cyclomatic Complexity (CC), which grows depending how many branches and loops that are inside the code. This metrics have been calculated using Radon[1] for Python codes and Shellmetrics[2] for the bash script. In the table, we can see that the LLoC used in the bash version is 57% larger than the PyCOMPSs version and 3 times more complex. Moreover, it includes functionalities that were not available in the bash script, such as the partial results generation which was available for the first time in the eFlows4HPC PTF workflow.

In addition to the workflow programmability, the porting to PyCOMPSs has also brought some performance benefits. In the original version, the post-processing of the results was done locally after all executions were completed and files transferred from the HPC resources. In the eFlows4HPC workflow, the post-processing of the simulation output is also performed in the HPC center. The COMPSs runtime detects the dependencies between the simulations and the post-processing tasks so that, immediately after the completion of the GPU simulations, the post-processing tasks depending on these simulation outputs, can run in parallel with subsequent simulations (using the available CPUs).

2.2 Deployment and portability of the Tsunami workflow

Prior to eFlows4HPC, all components of the workflow were installed manually, and its execution steered interactively by a user from a terminal. The eFlows4HPC PTF workflow has been integrated with the HPCWaaS interface. The PyCOMPSs workflow code is associated with a YAML file describing the software requirements, which is subsequently used by the Container Image Creation service[3] to create the container for the Workflow, and a TOSCA description which is used to automate the deployment and execution of the workflow.

Figure 1 displays the TOSCA topology for the PTF workflow, describing the elements involved in the deployment and execution of the workflow in an HPC site. Dependencies between components indicate that a component requires some data that is described or generated by the execution of another component.

[1] https://radon.readthedocs.io/

[2] https://github.com/shellspec/shellmetrics

[3] https://github.com/eflows4hpc/image_creation

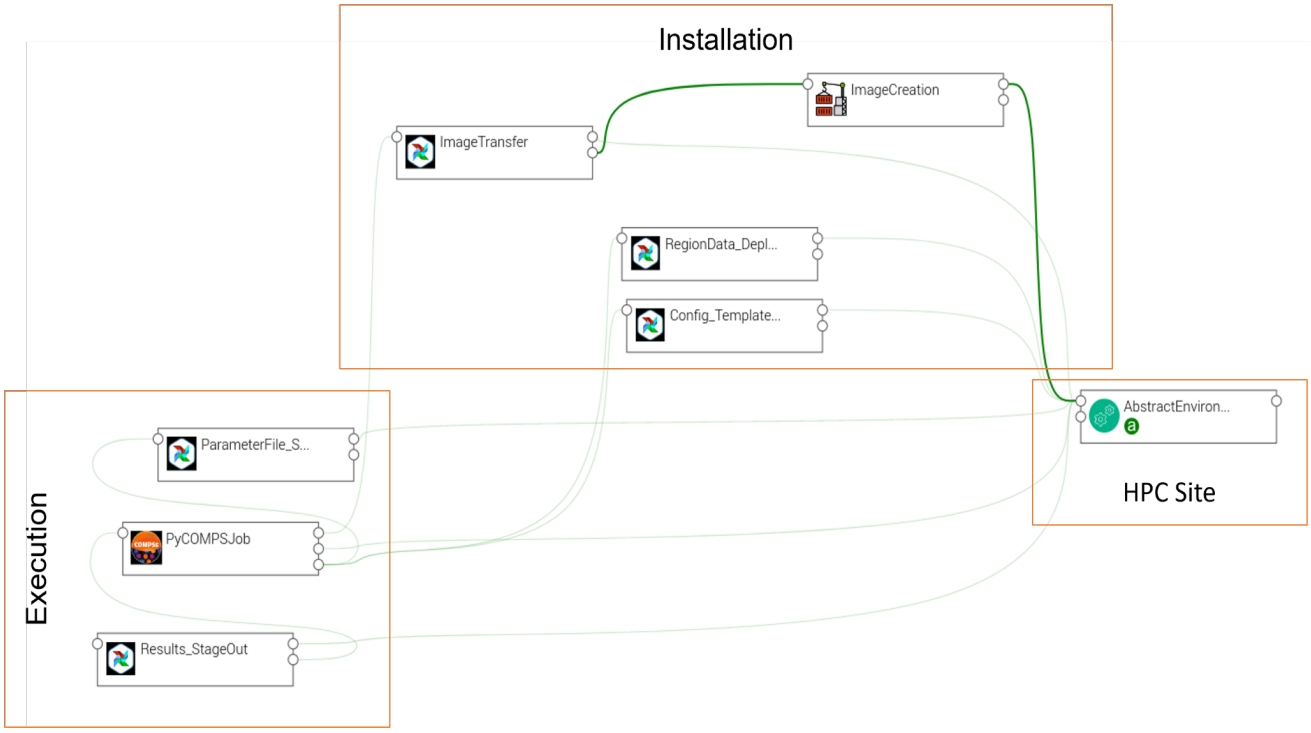

Figure 1. TOSCA Description for the PTF Workflow.

In this case, we have defined an AbstractEnvironment component which describes the properties of the HPC site (such as, for example, the login node address), the CPU architecture, and the MPI and GPU runtime types and versions. Then, for the installation phase, we have defined an Image Creation component which will request the creation of the image for the PTF workflow and target HPC site description, and different data pipelines which describe the transfers of the generated image, the regional data, and the configuration templates required by the PTF workflow.At execution time, we have defined three TOSCA components: one for the data pipeline that transfers the parameters file from the B2DROP to the HPC site, which is used to customise the workflow execution, another one to define the execution of the PyCOMPSs workflow in the HPC site, and a third one that describes the stage-out of the data generated by the workflow execution to the B2DROP repository. The dataset used in the workflow is described in the Data Catalog, where we have indicated the location of the data, the access protocol, and other metadata to follow the FAIR principles. To deploy the PTF workflow, developers select the HPC resource and dataset to be used in each deployment, and the TOSCA orchestrator (Yorc) executes the components required for the workflow installation in this site automating its deployment. This ensures that the same deployment can be performed for different sites or datasets

Table 3 compares the deployment and execution of the PTF workflow before and after eFlows4HPC. The comparisons are mostly more qualitative than quantitative, with an automation of tasks which were previously executed manually. With the eFlows4HPC HPCWaaS methodology and tools, all the deployment and execution steps are automatically executed ensuring reproducibility and portability.

Table 3. Comparison of deployment and execution processes for the PTF workflow before and after the eFlows4HPC project.

Phase	Step	Original	eFlows4HPC	
Deployment	Software Deployment	Manual transfer of sources and on-site compilation.	Automated with Yorc, CIC and DLS. Deployed as containers.	**Time** Image Creation: 5h/12h[4] Transfer: 6 mins
	Data Deployment	Manual transfer. Not FAIR-compliant.	Automated with Yorc and DLS. Data sources are obtained from the Data Catalog with FAIR compliance.[5]	11 seconds
Execution	Stage-in	Programmed in the bash script	Automated with Yorc and DLS.	4 seconds
	Execution	Manually invoked from the user's terminal.	Automated with Yorc and PyCOMPSs. Offered as service by HPCWaaS.	See performance results (next section)
	Transfers between steps	Programmed in the bash script.	Transparently managed by PyCOMPSs	Included in Execution
	Stage-out	Performed locally. Not FAIR-compliant.	Automated with Yorc and DLS. Uploaded to a B2DROP repository.	29 seconds

2.3 Performance of the Tsunami workflow

Strong and weak scaling experiments have been executed to validate the performance of the PTF workflow. In the strong scaling experiment, we measure the speed-up of the workflow execution when run with more resources, while in the weak scaling, we measure the relative efficiency of the workflow when performing larger executions.

[4] First time corresponds to creating the image in the same platform as the target, second time corresponds to the case when the target platform is different to the compilation one and emulation is used.

[5] Description of the dataset in the catalogue.
https://datacatalogue.eflows4hpc.eu/storage.html?type=dataset&oid=cd3f063f-c22a-428d-a731-d5428f938f9c

In the strong scaling experiment, we configured the workflow to evaluate 640 scenarios for the "2003_0521_boumardes" event[6]. For each scenario, a Tsunami-HySEA simulation is performed calculating 4 hours of wave propagation. The hazard curves and other post-processing operations are calculated every 20 finished simulations. This workflow is executed with a different number of nodes in the BSC CTE-Power9 machine, where each node includes 4 GPUs. We have used up to 16 nodes (64 GPUs), the maximum available in the queues accessible to us.

Table 4 and Figure 2 show the execution times and speed-up obtained for the different executions of the strong scaling experiment.

Table 4. Strong scaling results for PTF workflow in the CTE-Power9 machine. The relative speed up is the percentage of the ideal speed-up achieved, i.e. #nodes(speed-up)/2.*

Scenarios	Nodes	GPUs	Time	Speed-up	Relative speed-up
640	2	8	19.003	1	-
640	4	16	9.705	1,96	98.0%
640	8	32	4.917	3,86	96.5%
640	16	64	2.669	7,12	89.0%

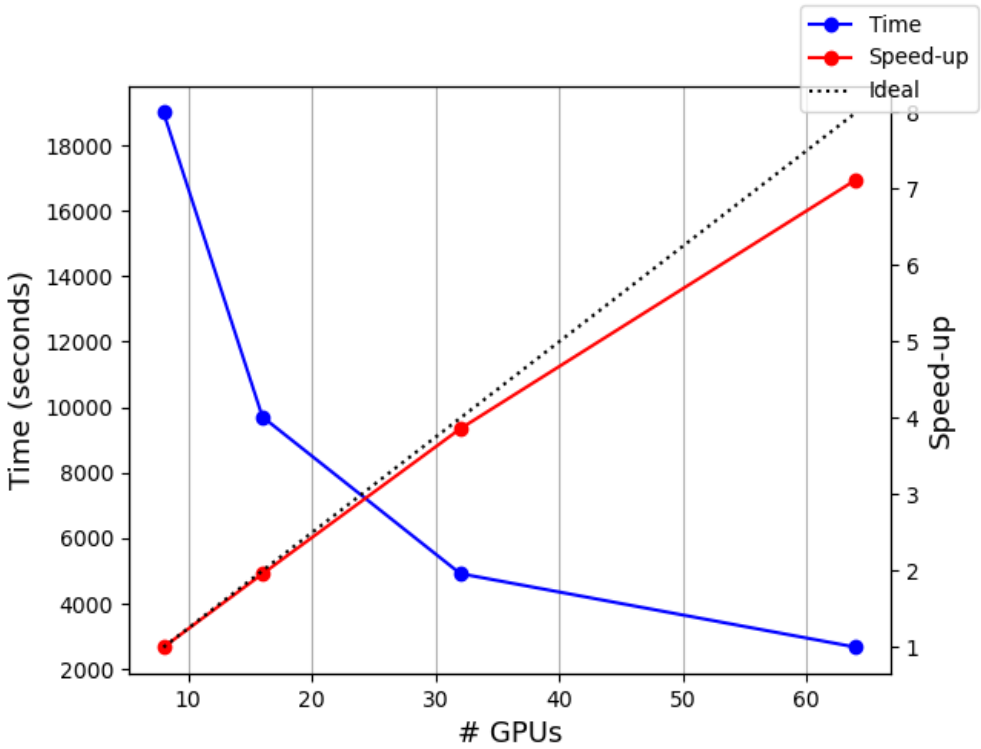

Figure 2. Strong scaling results for the PTF workflow in the CTE-Power9 machine. 640 tsunami scenarios were computed in each run using 2, 4, 6, and 8 nodes (8, 16, 32, and 64 GPUs).

[6] A magnitude 6.8 tsunamigenic earthquake on the Northern coast of Algeria on May 21, 2003. https://en.wikipedia.org/wiki/2003_Boumerd%C3%A8s_earthquake

The speed-up is computed taking as baseline the executions with 2 nodes. So, for example, the maximum speed-up achievable for 16 nodes in ideal conditions would be 8. In this case, we obtained a speed-up of 7,12 which is very close to the ideal. The difference is mainly due to the initial sequential part (the generation of ensembles and preparation of the simulation) that consumes proportionally more time as the parallel parts of the workflow are distributed between more resources.

In the weak scaling experiment, we kept the number of the HySEA simulations per node constant but increased the number of scenarios proportionally with the allocated computing resources. We would ideally see that the workflow execution time stays constant. Table 5 and Figure 3 show the time taken for different workflow execution configurations and the efficiency results. The efficiency is computed by comparing the time of each execution to the baseline run (the 2-node execution). We see that the execution time grows slightly but with an efficiency higher than 96%.

Table 5. Workflow execution configurations and execution times obtained for the weak scaling experiment of the PTF workflow.

Scenarios	Nodes	GPUs	Time	Efficiency
80	2	8	2.550	1
160	4	16	2.575	0,99
320	8	32	2.605	0,98
640	16	64	2.669	0,96

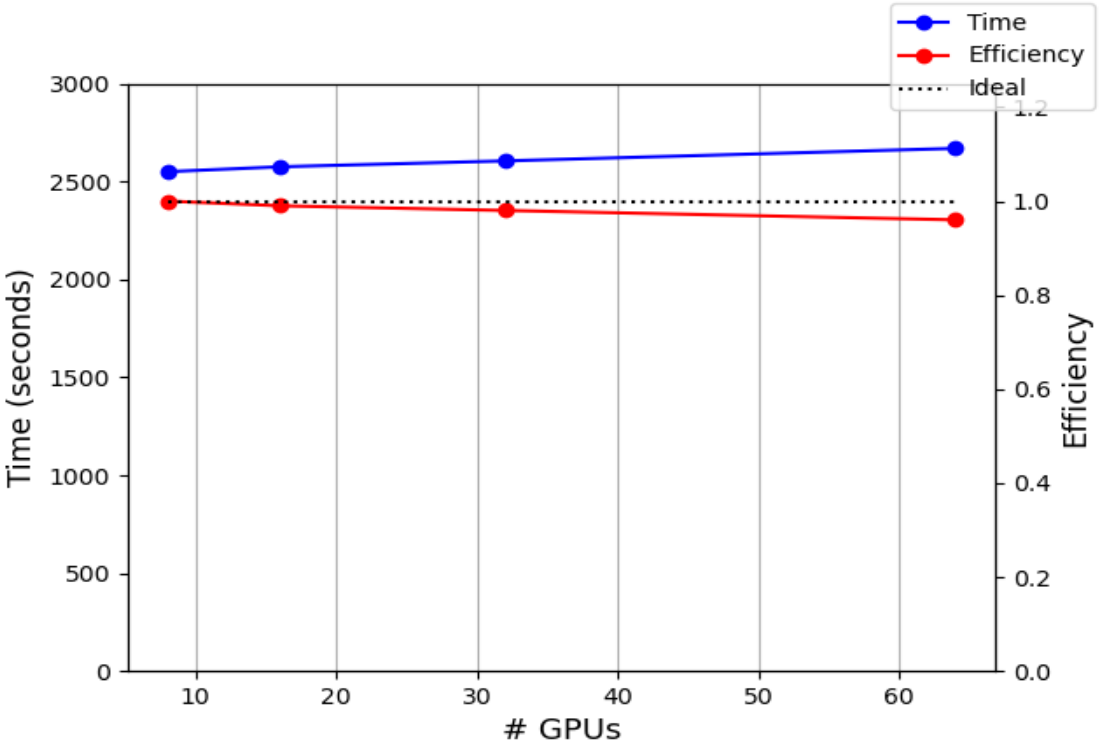

Figure 3. Weak Scaling results for the PTF workflow in the CTE-Power9 machine. The four executions displayed were computed on 2, 4, 8, and 16 nodes (8, 16, 32, and 64 GPUs) with a constant computational burden of 40 tsunami simulations per node.

3 Validation of Use Case Earthquakes

3.1 UCIS4EQ

The Urgent Computing Integrated Services for Earthquakes (UCIS4EQ) is a novel HPC-based urgent seismic simulation workflow (WF). It became a candidate workflow in the eFlows4HPC project's Pillar III with the objective to improve its maturity level through the integration of different components proposed in the Project's software stack.

UCIS4EQ was proposed as a collection of virtualized services. At the beginning of the eFlows4HPC project, UCIS4EQ had a low maturity level and was orchestrated only by a sequential workflow manager (WM) emulator. Several execution requirements were described in D6.1 in order to further its development as a workflow and to improve both its capabilities and maturity.

The software stack components integrated in UCIS4EQ are well described in the deliverable 6.4, and in the project documentation https://eflows4hpc.readthedocs.io/en/latest/Sections/0_Intro.html.

The selected components of the eFlows4HPC software stack are key instruments in fulfilling the execution and deployment requirements of the UCIS4EQ workflow. A summary description of those components follows:

- **PyCOMPSs** is the most important technology used from the software stack to increase the maturity development level of UCIS4EQ in the project. It is implemented in UCIS4EQ as a major run-time component designed to orchestrate tasks in a distributed execution on a multi-node HPC cluster. This integration implied the development of new functionalities in PyCOMPSs, in particular the @http decorator that allows developers to define a task that performs HTTP requests. The PyCOMPSs integration as a workflow orchestrator endows UCIS4EQ with execution robustness, infrastructure interoperability, adaptivity, dynamicity, fault tolerance, and scalability (https://github.com/eflows4hpc/compss).
- **dislib** was selected from the *Data Analytics and Machine Learning library catalogue* to manage the training, testing and inference tasks in the proposed building block #8 (see Fig. 5.3 at Section 5). dislib has the capability to manage a large set of data through the inner distributed functionality, a feature that has proved crucial for our application (see Section 3.2 for more details and the project repository https://github.com/eflows4hpc/dislib).
- **Ystia orchestrator** based on the TOSCA standard has been integrated in the HPC part of UCIS4EQ to adopt the HPCWaaS methodology. (https://github.com/eflows4hpc/yorc)
- **DLS** Data Logistic Service is a tool to define, schedule, and execute in a reproducible way data pipelines. It collects and integrates distributed data before they are used for processing (https://github.com/eflows4hpc/data-logistics-service). In UCIS4EQ the data required in the HPC and the results are moved through a DLS pipeline. Certain TOSCA components of UCIS4EQ (D6.4) interact with the DLS.

This deliverable also reports a new functionality that confers malleability to UCIS4EQ, in that simulations can be cancelled or launched once the WF is activated. In particular, this activity is related to the Building Block (BB) #5, named "Event Assessment" (Fig. 5.3), and consists of two steps as exhibited in Figure 4.

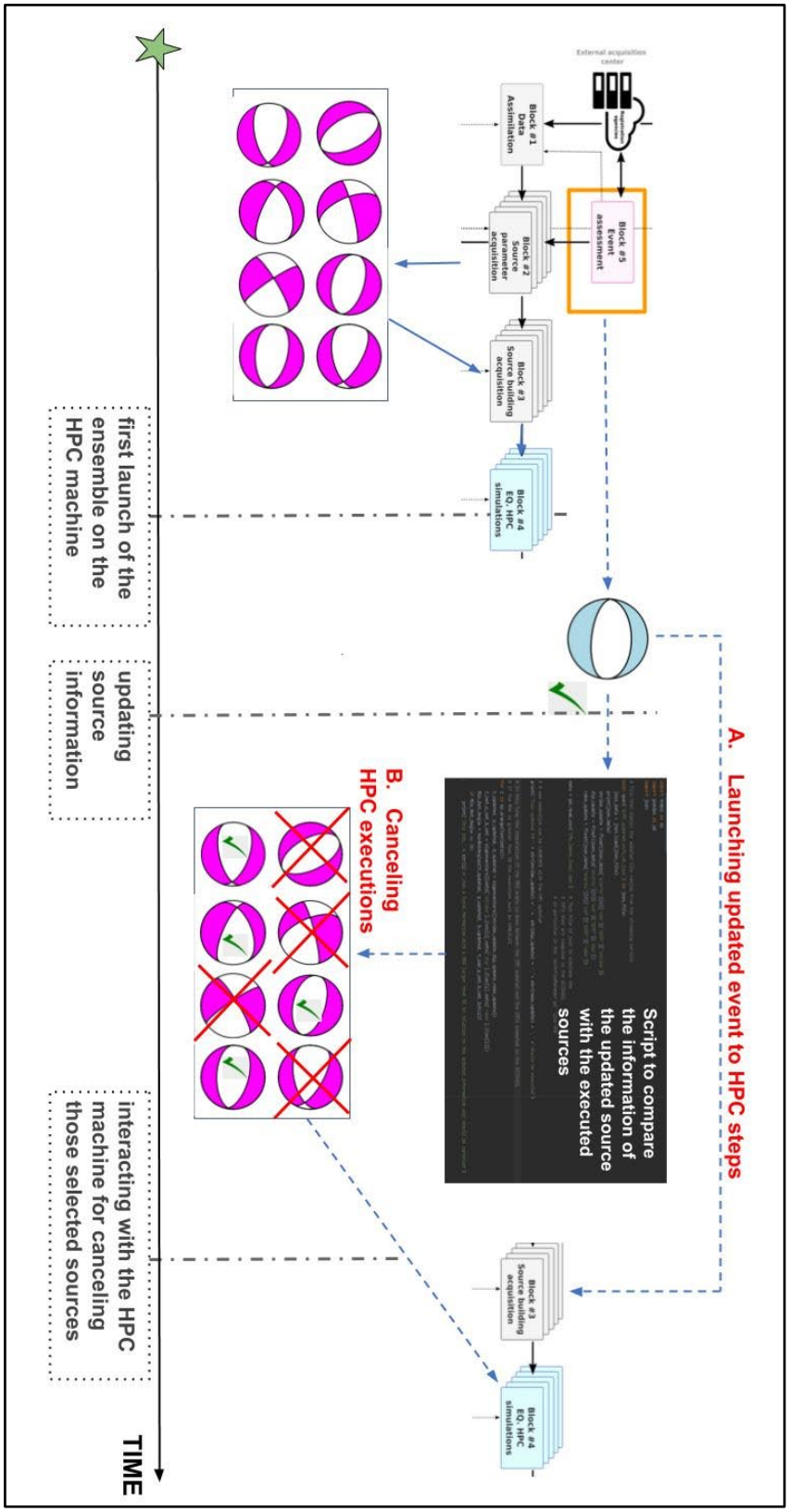

Figure 4. Schematic description of the malleability activity carried out in UCIS4EQ. This activity is related to the BB#5 shown in the orange box. The timeline in this figure indicates the WF execution starting at the earthquake (green star). The information is updated from data coming from external agencies. The updated source is assimilated in the WF continuing the execution. Moreover an intermediate script decides which sources previously launched should be cancelled.

A. **Staging/launching updated event for HPC execution:** In this activity, information on source parameters coming from external agencies are updated in BB#5. Event data is prepared, and a single event execution launched. To provide this functionality, a modification of the WF is incorporated to prepare the data for only one event without the Ensemble services that employ statistical and/or clustering sources augmentation.

B. **Cancelling HPC executions:** This functionality confers malleability to UCIS4EQ in that it permits the fully automatic cancellation of current executions on an HPC infrastructure. To specify which sources must be cancelled, the new source and the ensemble are compared based on the angles between the source's geometries. Events to be cancelled are selected and their source ID is processed to identify the corresponding job.

This malleability feature has been introduced in the workflow with the changes depicted in the code snippet of Figure 5. For each event, we have grouped the simulation tasks according to the source's geometries (*cmts*) and we have created a data stream and the *monitor_event* task which will run in parallel with the simulation tasks. This task compares the source's geometries of alert updates for a given event and the ones of the running simulations as mentioned above. Based on this comparison, it publishes the actions derived from every update (new simulations, or cancellations) in the stream. After launching the initial simulations, the main code of the workflow executes the *update_simulations* function which listens to the stream and applies the actions communicated through it, launching the simulations of the updated geometries and cancelling the task groups for those simulations whose geometries are invalidated by the updates.

```
jorgee@bsccs132:...ode/components/scc 134x44

            precmt = build_cmt_input(eid, region, resources, setup)
            all_results = []
            alert_num = 0
            ods = ObjectDistroStream(alias="updates")
            for alert in event['alerts']:
                cmts = calculate_cmt(alert, eid, region, precmt)
                cmts = compss_wait_on(cmts)
                # Monitor new events
                monitor_event(alert_num, cmts, ods)
                for cmt in cmts.keys():
                    with TaskGroup("alert_" + str(alert_num) + "_" + str(cmt), False):
                        for slip in range(1, gpsetup['trials']+1):
                            path = basename + "/trial_" + ".".join([cmt, "slip"+str(slip)])
                            all_results.append(launch_simulation(eid, alert, path, cmts[cmt], region, gpsetup, resources, ensemble))
                alert_num = alert_num + 1
            # Cancel or launch new simulation according to event updates
            update_simulations(ods, eid, event['alerts'], basename, region, gpsetup, resources, ensemble, all_results)
        result = launch_post_swarm(eid, region, basename, resources, all_results)
        result = compss_wait_on(result)
        eid = set_event_state(eid, "SUCCESS")
        return jsonify(result = "Event with UUID " + str(body['uuid']), response = 201)

@task(obs=STREAM_OUT)
def monitor_event(alert,cmts,obs):
    updates = process_update(cmts, alert)
    for update in updates:
        obs.publish(update)
    obs.close()

def update_simulations(ods, eid, alerts, basename, region, gpsetup, resources, ensemble, all_results):
    while not ods.is_closed():
        alert_updates = ods.poll(timeout=10000)
        for update in alert_updates:
            if (update.type == 'NEW'):
                with TaskGroup("alert_" + str(update.alert) + "_" + update.source, False):
                    for slip in range(1, gpsetup['trials']+1):
                        path = basename + "/trial_" + ".".join([update.source, "slip"+str(slip)])
                        all_results.append(launch_simulation(eid, alerts[update.alert], path, update.cmt, region, gpsetup, resources,
ensemble))
            elif (update.type == 'CANCEL'):
                compss_cancel_group("alert_" + str(update.alert) + "_" + update.source)
```

Figure 5. Changes in the UCIS4EQ workflow to support malleability. Two new functions have been defined and read squares highlight the changes in the main code of the workflow.

3.1.1 UCIS4EQ requirements and pillar III metrics

At this final implementation stage, we augmented UCIS4EQ with new features, and validated against initial requirements and metrics. Table 6 summarises the workflow requirements and the degree of completion reached at this validation stage.

Table 6. UCIS4EQ requirements fulfilment analysis at the final stage of the project.

Name	Priority	Status	Comment
Data replication	must	implemented	Currently the data for the different test cases are replicated at two distinct HPC centres, MareNostrum 4 (BSC, Spain) and Piz Daint (ETHZ, Switzerland). Static data used by UCIS4EQ is also replicated on the B2DROP and B2SHARE services: (https://b2share.eudat.eu/records/fa260bf13a8f480aae66db4940c95f83).
Execution Robustness	must	Implemented	The PyCOMPSs workflow manager ensures fault tolerance during the workflow execution including checkpoints or retries.
Infrastructure interoperability	must	Implemented	Infrastructure interoperability is made possible by the deployment system and thanks to the CIC and via the introduction of PyCOMPSs that orchestrates computations performed on different infrastructures (HPC clusters and external servers).
Integrated workflow manager	must	Implemented	PyCOMPSs is fully integrated within UCIS4EQ as is described in D6.4
Integration with permanent storage	must	Implemented	Currently the storage of the data required by the workflow is split between repositories at HPC centers for large files (MN4 and Piz Daint), as well as B2DROP (replicated on B2SHARE) for lighter configuration files. The final results of the workflow are automatically uploaded to B2DROP, while the intermittent results (such as full simulation outputs) remain on the HPC cluster. Integration of storage in long-term repositories is currently analyzed to be done B2SHARE
Inference with online/offline ML models	must	Implemented	MLESmap provides ML models that can be used in UCIS4EQ within Building Block 8. The ML models are prepared "offline" (that is, not in an urgent manner) and must be ready for an online application with rapid queries. The current test is being implemented in Iceland region as use case, but remains a non-integrated feature of UCIS4EQ, i.e. standalone, within the framework of the project.
Urgent computing access	must	Proposed	A first attempt to provide a protocol to enable the Urgent computing access mode in Tier-0 and Tier-1 machines isdelivered in this D6.6.
Streaming Data Source	must	Implemented	Acquisition of real-time streaming data sources should be developed in Building Block 5. A first algorithm was implemented, and calls for better integration in the workflow.
Data accessibility	should	Implemented	Data accessibility is provided by the Data Logistic Service through a pipeline that moves the data required to the HPC and the results. In this final iteration, the workflow relies on the B2DROP repository. This requirement is also important for a correct deployment of the workflow in other HPC Centers.

Workflow malleability	should	Implemented	Malleability is a characteristic inherent to PyCOMPSs and inherited by the HPC components of UCIS4EQ where it is implemented. It was tested to showcase that the resource management requirement is malleable.
Portability and Reusability	may	Partially implemented	The HPC workflow can now be safely deployed by means of TOSCA. This makes the workflow portable to a variety of HPC systems. Workflow components are containerized to permit their substitution or reuse. Implementation of this requirement is deemed partial due the fact that SALVUS, being distributed as precompiled binaries, cannot be compiled to a container image creation service. It is only supported for designated architectures, impeding its deployment on the premises of any HPC Center.
DA integration	may	Not integrated	The UCIS4EQ post-processing stage requires the management of large HDF5 output files obtained from each simulation of an ensemble of sources, the objective being to collect final results in a single file. The management of such HDF5 files is not provided by any data analytics tool in the project. We therefore resort to Python scripts to carry out this task.

At the beginning of the Project (D6.1), we suggested three Pillar III specific metrics to quantify the QoS of the workflows at Project's end. Those metrics are included in Table 7.

Table 7. Pillar III metrics defined at D6.1. The final measure of advancement of the Project is described in column 3.

Specific metrics for WP6	Description	Measure
Response time (RT)	End-to-solution time constraints in an urgent computing context. This metric is defined as the wall-clock time measured from the event's reception until the first valid solution for the event is delivered to the stakeholder, e.g. a civil protection agencies. For the metrics to serve in comparing different workflow instances, ie. execution for events using more or less resources, RT must be normalised with respect to the computational resources used.	RT was only partially tested in an urgent computing context due to the lack of an urgent computing queue. RT is measured for Scenario 2 described in Table 10. To compare RT between two workflow executions that consume different computing resources, we set two executions with a maximum of 7680 CPUs and 10752 CPUs each. We obtain RT=9.27h and RT=1.21h respectively. After normalization by the number of utilized CPUs we obtain: RT_N=0.072 minutes/CPU for 7680 CPUs RT_N=0.006 minutes/CPU for 10752 CPUs
Urgent Assignment of Resources	Time to access necessary job execution resources in an urgent computing situation. The metric quantifies the QoS in HPC facilities that provide UC services. It is a central measure of whether the adopted Urgent Computing policies are adequate for an UC execution.	Not measured in a production environment for lack of an operational UC policy.
Results Uncertainty Quantification	The RUQ metric is proposed as a measure of the uncertainty associated with the UCIS4EQ workflow results.	In the evaluation of the service, RUQ is defined as $RUQ = \frac{1}{N}\sum_{i=1}^{N} \lvert\hat{s}_i - o_i\rvert$ where $\hat{s}_i$ is the average value of the source ensemble results, and o_i is the observation at each available station

	RUQ is implemented in a workflow execution context where it is crucial to constrain and reduce the uncertainty of provided impact estimates. The uncertainties must be calibrated and assigned, among others, via the available GMPEs in the study region, observations from monitoring systems, information of past earthquakes, and results from ML algorithms.	*i, i=1,...N* stations. The ideal value of this metric is zero and an example follows: For two Mediterranean earthquakes of different intensities (M7.0: Samos-Izmir and M6.6: Kos Bodrum) as showcased in D6.2, we compute RUQ applied to the maximum Peak Ground Acceleration in the horizontal components (PGA_{max_h}) and to the maximum Spectral Acceleration at 0.3s period in the horizontal components ($SA0.3_{max\_h}$). Obtained results are: **M7.0: Samos-Izmir** RUQ(PGA_{max_h}) = 27.5cm/s², and RUQ(SA 0.3_{max_h}) = 74.2 cm/s² considering 24 stations. **M6.6: Kos Bodrum** RUQ(PGA_{max_h}) = 31.2cm/s², and RUQ(SA 0.3_{max_h}) = 75.4 cm/s² considering 11 stations These results are a first attempt at quantifying result uncertainty using a metric. We expect to be able to further refine the RUQ definition and maximum allowed constraints in future efforts, beyond this Project.

3.1.2 Programmability of the UCIS4EQ workflow

Prior to the eFlows4HPC Project, an initial version of the UCIS4EQ workflow was developed as part of the CHEESE project (viz. Appendix D). That early version consisted of a set of microservices, each specialised in the execution of distinct computational tasks. Some of these microservices were tasked with submitting jobs to the HPC infrastructure. Another was tasked with invoking yet another microservice every time a new seismic event of interest was detected. Altogether a large part of the code was dedicated to processing service calls, and to managing the asynchronicity of service calls for HPC job submissions. The first version implemented in the eFlows4HPC Project (viz. Appendix E) was a port of this orchestration service to PyCOMPSs, keeping the rest of microservices but performing the service calls as tasks implemented with the @http decorator. As shown in Table 8, the code required to implement the same functionality was reduced from 114 LLoC to 87 and the Cyclomatic Complexity was reduced from 9 to 5. The original code version used 31% more LLoC and its code was 80% more complex.

Table 8. Programmability metrics for the UCIS4EQ workflow versions and its comparison with the original version.

Version	LLoC	Cyclomatic Complexity
Original	114	9
PyCOMPSs	87	5
PyCOMPSs + HPC	70 (23)	5 (1)

A second version of the workflow focused on reducing the number of submitted HPC jobs and on facilitating the deployment with the HPCWaaS. In this case, we have split the workflow in two parts: the event management part that remains in the microservice; and the HPC part that is ported to PyCOMPSs and deployed and executed with the HPCWaaS methodology. From the programming perspective the microservice code has been reduced from 87 to 70 LLoC and the

HPC workflow requires 23 LLoC. However, this version does not require the implementation of the submission microservices which required a larger implementation of more than 300 LLoC each.

In this project, we implemented a 3rd version, which includes fault tolerance, management of the streams and malleability, but it is not comparable in terms of functionality and code with the original one.

3.1.3 Deployment and execution of the UCIS4EQ HPC workflow with the HPCWaaS

As mentioned in the previous section, the initial version of the UCIS4EQ already automated the execution and the data transfers generated during the execution. However, the deployment in the HPC site was done manually. In eFlows4HPC, some steps of the flow have automated the deployment and porting the HPC executions to the HPCWaaS model using the TOSCA description described in D6.4 Section 3.1.2.

Table 9. Comparison of deployment and execution processes for the UCIS4EQ workflow before and after the eFlows4HPC project.

Phase	Step	Original	eFlows4HPC	
				Time
Deployment	Software Deployment	Manual transfer of binaries. Manual installation of the required Python modules	Limited deployment as containers. Only compatible with Salvus binary options. Transferred by Yorc DLS.	Image Creation: 8h Transfer: 418 seconds
	Data Deployment	Manual transfer and no FAIR	Automated with Yorc and DLS. Data sources are obtained from Data Catalog compliant with FAIR.	971 seconds
Execution	Stage-in	Programmed inside the microservices	Automated with Yorc and DLS.	11 seconds
	Execution	Programmed inside the microservices	Automated with Yorc and PyCOMPSs. Offered as service by HPCWaaS	See performance results (next section)
	Stage-out	Programmed inside the microservices	Automated with Yorc and DLS. Uploaded to a B2DROP repository.	7 seconds

Table 9 shows the comparison of the status of the workflow deployment and execution before and after eFlows4HPC. Regarding the software deployment, the workflow uses Salvus to perform the Seismic simulations. It is a commercial software which is distributed as binaries and also contains a Python module that has to be installed as a Python wheel. This distribution mode has prevented

us from having a CIC compatible version. We can create and execute the software in containers as well as automating the transfer to the HPC, but it can be only created for the architectures available at the Salvus download service.

Regarding the data deployment, we have identified which data is common for all the executions (maps and other regional data), we have stored them in the UCIS4EQ B2DROP repository and registered to the Data Catalog. Every time the HPC part of the workflow has to be ported or reproduced in a new HPC site, the required data will be transferred together to the new HPC cluster in the deployment phase.

The execution was already automated in the implementation of the microservices in the original version, so there are no major differences in terms of functionality. However, the new implementation improves compatibility and reduces the necessity of extra code developments for a new HPC site. As previously outlined, in eFlows4HPC we merged all HPC-executed tasks in a single workflow implemented with PyCOMPSs and offered as a Service with the HPCWaaS execution API. We have therefore addressed the main drawback of the previous microservices solution: their implementation had to be adapted for every HPC site to ensure compatibility with the access protocol and queue system. With the new implementation of the single HPC workflow, the microservices responsible for submitting the jobs are not required anymore. The adaptation to the HPC sites is now done by Yorc and PyCOMPSs, so no extra effort is required from the developer.

3.1.4 Performance of the UCIS4EQ workflow

The M7.1 Puebla earthquake occurred on the 19th of September 2017. It was deadly. That event was proposed as a representative use case for earthquakes in D6.2 and described in D6.4. The same event was used in Task 6.5 (M30-M36) where UCIS4EQ ran in a production-like environment. Of note is the fact that the National Seismological Service (NSS) of Mexico explicitly supported the UCIS4EQ tests and helped us analyse the WF results in a first proof of concept (Monterrubio-Velasco et al., 2024a). To stir the interest of an operational seismological service in a workflow resulting from this Project was an unequivocal encouragement to us. Annex E contains the letter of support sent to us by the NSS of México.

We prepared different scenarios to map out the performance of UCIS4EQ executions for different seismic events: (i) the 2017.09.19 M7.1 Puebla earthquake (described in D6.2), (ii) the 2022.09.19 M7.7 Michoacán event proposed by the NSS of México. As shown in Table 10, the differing configurations of those events are described, in particular the maximum frequency used, the number of sources per ensemble, and the domain size (see Figure 6 for the spatial domain size).

Table 10. Configuration of the 4 scenarios used for use-case validation. Each scenario describes the domain size, frequency and number of representative sources used per ensemble. Representative sources are obtained from a clustering study. Figure 6 shows the domain sizes.

Scenario	Earthquake	Domain size [km]	Frequency [Hz]	Number of sources per ensemble
1	M7.1 Puebla (blue domain Fig. 3.3)	215 x 160 x 100	0.5 Hz	12

2	M7.1 Puebla (purple domain Fig. 3.3)	290 x 170 x 100	1.0 Hz	14
3	M7.7 Michoacán (orange domain Fig. 3.3)	115x150x 50	0.5 Hz	10
4		115x150x 50	1.5 Hz	10

Figure 6. Computational domain size of the Mexican earthquakes used to validate the UCIS4EQ workflow. The orange area corresponds to the M7.7 Michoacán event, and Blue and Magenta to the M7.1 Puebla earthquake. Both earthquakes occurred on September 19th in 2022 and 2017 respectively.

Those scenarios were used to test the performance metrics of the workflow. Strong scaling was applied to the most computationally demanding task of the UCIS4EQ workflow, i.e. the Salvus execution on the HPC infrastructure. Specifically, we used Scenario 2 of Table 10 with the largest domain size. Table 11 and Figure 7 depict the run time and speed-up considering the execution of only one source.

Table 11. Metrics of the HPC-Salvus execution in UCIS4EQ using scenario 2 as test case.

Nodes	CPUs	Time [min.]	Speed-up	Ideal
2	96	113,63	1,00	1
4	192	57,72	1,97	2
6	288	37,85	3,00	3
8	384	28,77	3,95	4
10	480	24,10	4,72	5
14	672	16,82	6,76	7
16	768	14,90	7,63	8

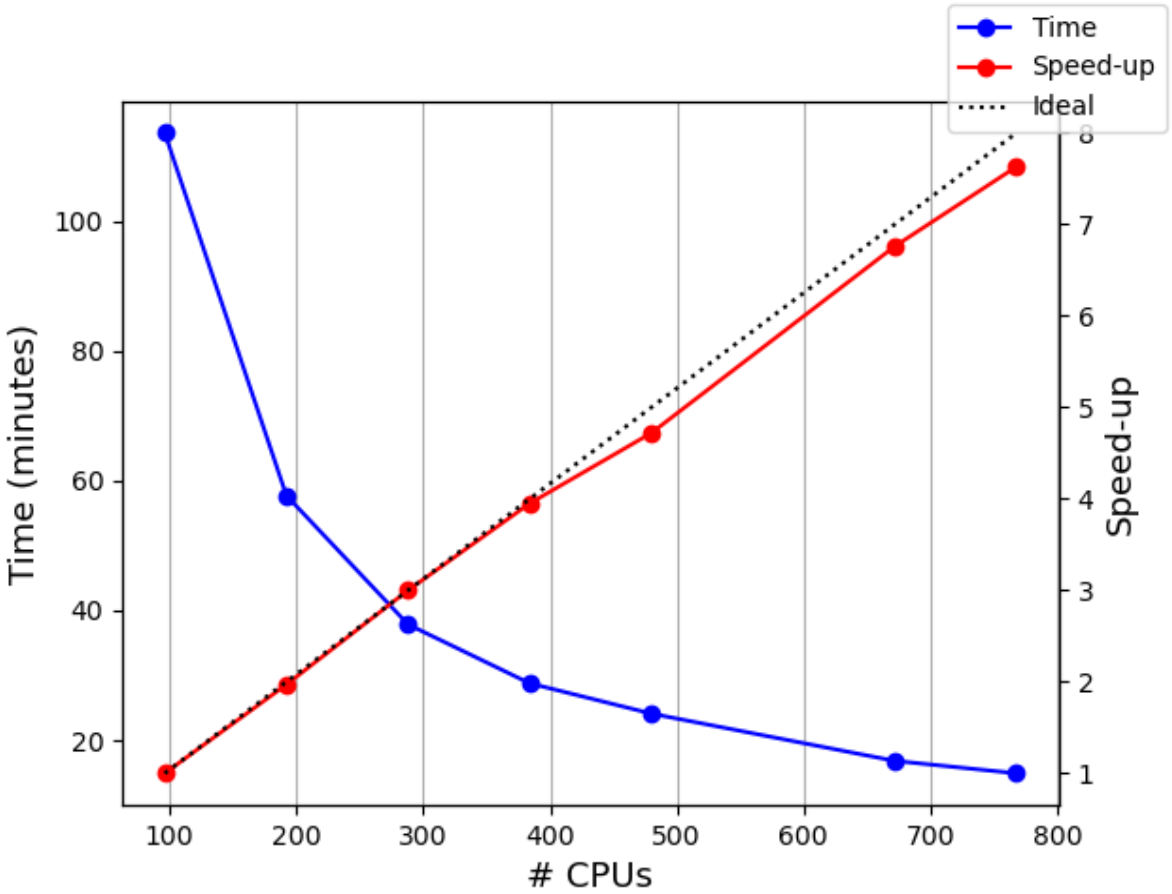

Figure 7. Run time and speed-up for Scenario 2 of Table 3.5. Workflow execution was only one source of the ensemble, and results shown are limited to the computationally most expensive task in the workflow, i.e. the HPC execution.

Since the parameters that influence the time-to-solution in the earthquake simulation are extremely non-linear, we compare the run-time obtained for the 4 scenarios described in Table 10. All scenarios are assigned the same number of computing resources, 10 Nodes and 480 CPUs, as shown in Table 12.

Table 12. Run-time computed for the four scenario configurations (Table 10) remaining the same computer resources of 10 Nodes and 480 CPU's..

Scenario (from Table 10)	Num Elements Salvus mesh	Salvus Time steps	Run-Time [minutes]
1	85021	33370	12
2	407346	70680	89
3	48048	100532	15
4	272427	200590	168

3.2 MLESmap

The Machine Learning Estimator for Ground Shaking Maps (MLESmap) harnesses the predictive capabilities of Machine Learning (ML) algorithms. Its training data consists of ground shaking accelerations obtained from physics-based seismic simulations. With MLESmap our intent is to provide ground shaking information within a few seconds following an earthquake. The inferred ground shaking values are quasi-real-time information that can be leveraged to explore uncertainties in the ground shaking values (p.e. PGA, SA) quickly and reliably.

In Pillar III, MLESmap was proposed as a novel methodology to provide offline-trained models for an online service defined in the UCIS4EQ workflow, specifically in Building Block 7. The planned interaction between UCIS4EQ and MLESmap workflows relies on the model repository that integrates the offline trained ML model for online use. At this stage, the MLESmap offline training workflow is implemented as a separate workflow and integrated to the HPCWaaS interface but the online inference phase is not integrated into UCIS4EQ. It is important to mention that MLESmap was incorporated in this project as an extra workflow for earthquake natural hazards and was not explicitly defined in the project proposal.

D6.4 summarises the components of the eFlows4HPC software stack used to improve the MLESmap workflow. The most important changes in MLESmap consist of a better programmability brought about by PyCOMPSs, and an improved execution thanks to the dislib library.

The execution of the MLESmap workflow has been implemented for South Iceland region as described in D6.4 and presented in Monterrubio-Velasco et al. (2024b)

3.2.1 Programmability of the MLESMap workflow

The MLESmap workflow has been implemented from scratch during the Project and there is no original version with which to compare it. Notwithstanding, we calculated the programmability metrics of both the code that implements the offline training as well as that of the code that performs inferences using dislib (Appendix F).

On the one hand, the offline training workflow uses the dislib's Grid Search algorithm to perform a hyperparameter search to find the configuration of the Random Forest Regressor model that better predicts the ground shaking values of a region. On the other hand, the inference workflow uses the dislib to load the Random Forest Regressor model found in the offline training and make the predictions.

Table 13 shows the metrics of the code programmed with dislib (dislib uses PyCOMPSs to parallelize and distribute the executions across computing nodes). Due to the simple dislib interface, the model selection workflow has been programmed with just 32 lines of code where we mainly load the dataset, scale it and create the parameter search. For the inference code, we exhibit similar results, where only 21 lines of code are necessary to load the models and event data, and to perform the inference. In both cases, the cyclomatic complexity is 1, as we do not require to program any loop or conditional.

Table 13. Programmability metrics for the MLESMap workflow.

Version	LLoC	Cyclomatic Complexity
Train	32	1
Inference	21	1

3.2.2 Deployment and execution of the MLESmap workflow

During this last period of the project, we have also ported the MLESmap offline training workflow to the HPCWaaS interface, so this workflow can be easily ported to other machines and run with different datasets. For this purpose, we have defined the TOSCA topology defined in Figure 8, and stored together with the software requirements and the PyCOMPSs workflows in the Workflow Registry[7]. In this topology, we have included the ImageCreation and ImageTransfer components to create the workflow container image and transfer it according to the HPC site features. This part defines the deployment part. For the execution phase, we have defined two TOSCA components to represent, the stage in of the training data set and the parameters to evaluate in the parameter tuning to the selected HPC site. Then, we have defined a PyCOMPSs Component which will perform the workflow execution in the HPC site. Finally, another TOSCA component is defined to perform the upload to the Model Repository.

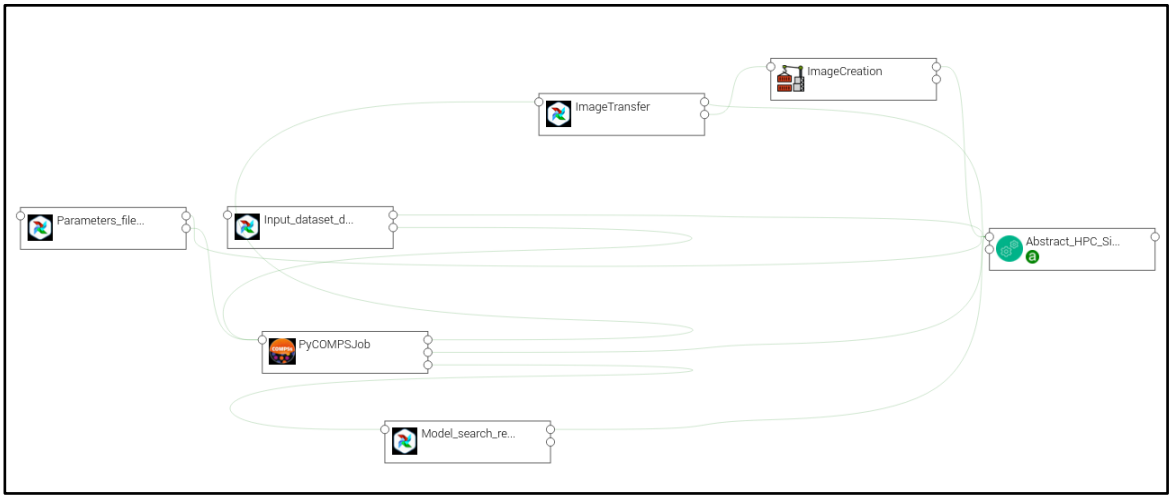

Figure 8. TOSCA Topology for the MLESmap workflow.

[7] https://github.com/eflows4hpc/workflow-registry/tree/main/Pillar_III/mlesmap

Table 14 shows the status of the different deployment and execution phases for the MLESmap workflow at the end of the project. As you can note, all the deployment and execution steps have been automated and the workflow execution is offered as a service using the HPCWaaS execution API.

Table 14. Deployment and execution processes for the MLESmap workflow after the eFlows4HPC project.

Phase	Step	eFlows4HPC	
Deployment	Software Deployment	Deployed as containers using the CIC DLS and Yorc.	**Time**
			Image Creation: 2,5h/5h Transfer: 305 seconds
Execution	Stage-in	Automated with Yorc and DLS.	Dataset: 167 seconds Parameters.json: 4 seconds
	Execution	Automated with Yorc and PyCOMPSs. Offered as service by HPCWaaS	See performance results (next section)
	Model	Automated with Yorc and DLS. Uploaded to the Model Repository	118 seconds

3.2.3 Performance of the MLESMap workflow

This section reports the performance evaluation of the MLESmap training workflow. This performance evaluation is divided into two parts. The first part evaluates the scalability of the training and parameter tuning workflow implemented with dislib on top of PyCOMPSs, and the second part measures the performance of the inference code.

Regarding the scalability evaluation, we performed two experiments: a strong scaling experiment, that measures the speed-up of running the parameter tuning workflow for a training dataset with 3 k-fold cross-validation, and an evaluation of 16 combinations of parameters of the Random forest regressor using the grid search algorithm.

Table 15. Strong scaling results for MLESmap workflow with 16 Grid Search(GS) combinations.

GS Combinations	Nodes	CPUs	Time[s]	Speed-up	Ideal
16	12	576	9.600	1	1
16	24	1152	4.802	2,00	2
16	48	2304	2.531	3,79	4
16	96	4608	1.472	6,52	8

16	192	9216	852	11,27	16
16	384	18432	744	12,90	32

Table 15 and Figure 9 show the results of the strong scaling experiments, where we can see a good speed-up until 192 nodes. The reason for the degradation for larger number of nodes is the size of the problem that limits the workflow parallelism. However, the algorithm can scale if we use larger datasets or combinations. To validate this assumption, we have performed a weak scaling experiment.

In the weak scaling experiment, we have increased the parallel workload of the algorithm in the same proportion as the number of computing resources and we have measured the computational efficiency. To do it, we have started with the evaluation of one Grid Search (GS) combination for 24 nodes, two GS combinations for 48 nodes and so on until sixteen GS combinations for 384 nodes are reached.

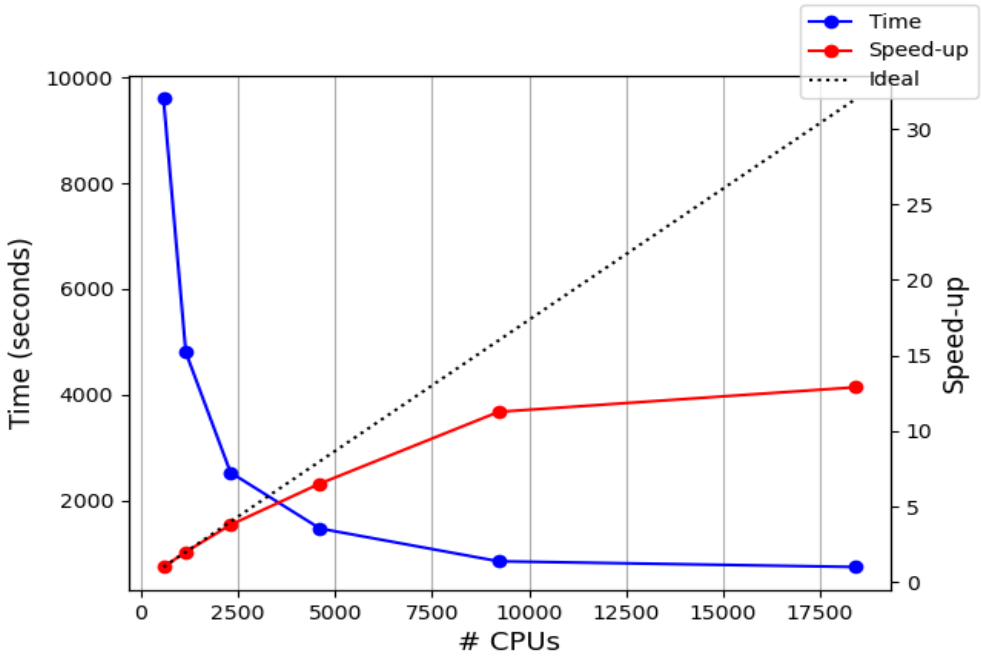

Figure 9. Execution times and speed-up vs number of CPUs for the MLESmap workflow with 16 Grid Search(GS) combinations.

Table 16 and Figure 10 show the results of the weak scaling experiments, where we can see that the efficiency is above 90% in all the cases.

Finally, we evaluated the inference time using dislib. Initially, we tried to perform a distributed inference as we did in the training. The times to obtain the inference can be found in the first column of Table 17. Despite much faster executions than the real simulation, we saw that the computation per task was very small, and the distribution of the computation was very inefficient. After this evaluation, we decided to execute the dislib script for the online inference in a sequential way. In this case, we measured extreme improvement in time-to-solution for the predicted time as depicted in the second column of Table 17. Incidentally a Python script calling the dislib library

can be run sequentially by just executing it with the standard Python interpreter, or can be executed in a distributed compute environment using the PyCOMPSs runtime.

Table 16. Weak scaling results for the MLESmap workflow fixing a load of 1 GS combination per 24 nodes.

GS Combinations	Nodes	CPUs	Time [s]	Efficiency
1	24	1152	696	1
2	48	2304	697	1,00
4	96	4608	702	0,99
8	192	9216	717	0,97
16	384	18432	744	0,94

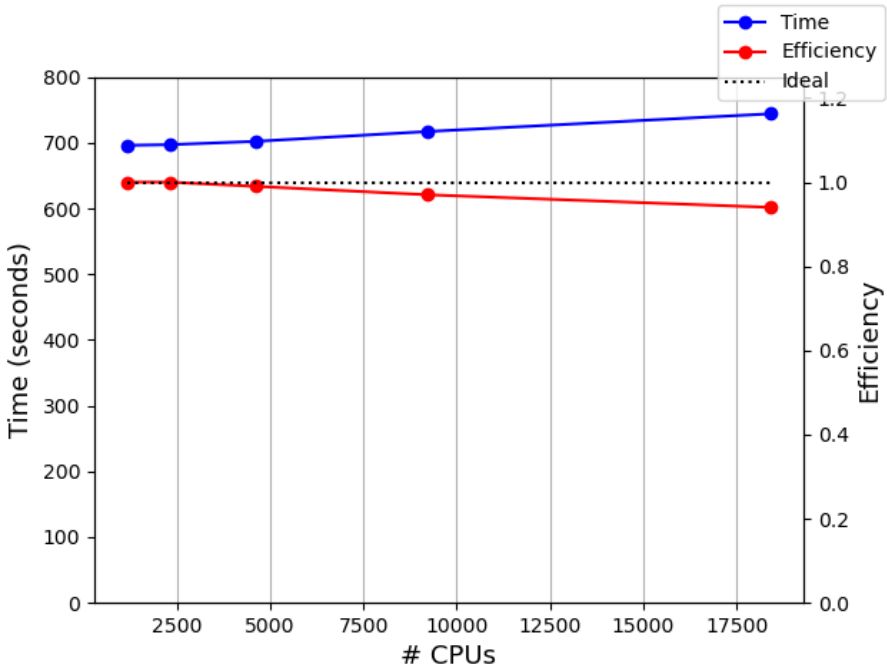

Figure 10. Execution times and efficiency vs number of CPUs for the MLESmap workflow with 1 GS combination per 24 nodes.

Table 17. Inference times with dislib.

dislib inferences on production	Time [s]	
	Before	Sequential
Load real data	0.06	0.0001
Transform data	74.75	0.0006

Load model	36.98	48.13
Predict	44.47	0.08

4 Urgent Computing Access Policy

4.1 Purpose

This policy serves to establish the guidelines for the fair and equitable use of urgent computing resources allocated within the HPC environment. Urgent computing involves the execution of time critical tasks that require immediate access to HPC resources.

The European Commission can play a vital role in promoting urgent computing and making it available to emergency responders across the European Union. By investing in urgent computing technologies, the Commission can help to ensure that Europe is better prepared to respond to crises. To do so, there is a need to have the financing model for the Urgent Computing infrastructure providers.

4.2 Background

This policy applies to all HPC users, actors. It encompasses the entire HPC facility, including compute nodes, storage systems, and network resources.

Urgent computing may be characterised by the following key features:

- Real-time or near-real-time performance: Urgent computing tasks must be completed within tight deadlines, often within seconds or minutes. This requires the use of high-performance computing resources and efficient algorithms to minimise execution times.
- High priority and preemptability: Urgent computing tasks are given priority over non-urgent tasks. This means that urgent jobs can preempt lower-priority jobs if necessary to ensure that they are completed on time. The lower-priority jobs may lose their current results!
- Resource isolation and dedicated access: Urgent computing often requires dedicated access to exclusive resources to ensure that there is no interference from other tasks. This can be achieved through special partitions, pools, or even dedicated hardware accelerators. Each of these solutions is expensive.
- Flexible job scheduling and management: Urgent computing systems must be able to handle unpredictable job arrivals and prioritise tasks effectively. This requires flexible job scheduling mechanisms and real-time monitoring capabilities to dynamically adjust resource allocation.

Several types of preemption have been used in the past [11] . The first type is kill/restart. In this type, preempting a job will kill the job and start it again at a later time. Once the job has been killed, it will return to the scheduler queue. Any partial results will be lost, and the time spent running the preempted job is wasted.

The second type of preemption is suspend/resume. In this type of preemption, a job is suspended and then resumed at a later time. All the processes associated with the job are stopped, but the state of each process is retained so that the job can continue when it is resumed. When preempting a job to execute an urgent job, the operating system will need to transfer the states to a swap memory space before the new job can start.

Based on the experience with the work on the production HPC environment (BSC, PSNC), the kill/restart preemption type is the first choice in the production environment. In general, where there is no possibility to predict and schedule the jobs with the defined memory requirements in order to allocate the resources for possible furthcoming urgent jobs. Some jobs can allocate entire node memory. In that scenario - swapping the memory image into the storage infrastructure takes the time which is not expected.

4.3 The policy proposal

Urgent computing (UC) refers to the execution of time-sensitive tasks with a high priority for immediate execution on a given computing infrastructure. This may include tasks that require immediate analysis of data, tasks that are needed to respond to real-time events, or tasks that must be completed by a certain deadline.

Ordinarily users of the computing infrastructure resources, or possibly the computing infrastructure itself, when preempted by an UC order are expected to suffer losses. Those losses range from seeing certain job execution starting times delayed, to having the currently executing job either aborted or stopped and restarted. In all cases, a sound UC policy should take into account collateral aspects detrimental to ordinary users or to contractual liens between the computing infrastructures and interested third parties.

As a result UC policies should include transparent compensatory provisions so both HPC computing infrastructures and their users affected such policies can be made whole.

One of the affected parties' compensation avenues to explore may be the adoption of a model similar to that for the financing of a fire service. The regular subscription model will help solve the issues of keeping the runtime environment ready as well as compensate for restarting computing jobs on the infrastructure.

5 Urgent Computing Operational Analysis

The applications and codes being used and enhanced during the project lifetime were tested in the pilot HPC environment in BSC and PSNC. Due to dynamic changes or prioritisation mechanisms to support urgent computing scenarios which can potentially impact the stability and reliability of the production environment, the application jobs were queried in the production queue (partition). We check that the statutes of resources' usage in HPC centres do not provide the information on the possible computational results' loose, as well as availability, limitations, and procedures for requesting urgent computing resources.

The pilot environment offered the queues for running the codes but without changing the priorities nor preemption policy. All evaluation results were obtained in the 'standard' way of acquiring resources on the system. The execution times (Walltime) were collected for different problem sizes. The deliverable presents the efficiency of codes before and after the work carried out in the project.

The evaluation on SLURM configuration for urgent computing mode was done in PSNC during the planned system maintenance. All users' jobs were affected in the production environment (killed) and urgent code (SALVUS) was placed and run. The time expected for the initialisation of the job did not exceed 1 minute. In the chapter we present the procedures for running the core parts of the applications in the urgent computing mode. The measured code runtimes for different data sets and resources' amounts have been presented.

5.1 Tsunami

This section addresses the general description of the application components designed to operate on High-Performance Computing (HPC) systems when deployed in urgent computing mode for a tsunami event. In the event of a potential tsunami, a structured operational protocol for urgent computing comes into play. This protocol orchestrates systematic steps to be followed when simulations are deemed necessary to assist early warning services. The protocol encompasses early detection mechanisms, criteria for triggering simulations based on detected events, and the swift allocation of HPC resources to execute an ensemble of high-fidelity simulations to produce a probabilistic forecast for the potential tsunami by using the PTF/FTRT workflow.

It involves the integration of consolidated and real-time data feeds, including real-time estimations of the seismic source's geometry, to enhance the accuracy of forecasts. Long-term and historical data (e.g. accurate topo-bathymetry, seismicity models) is also leveraged to bolster the reliability of simulations. The dynamic scaling of computational resources based on the urgency and scale of the simulated event is paramount, ensuring that the allocated resources align with the evolving demands of the situation.

The analysis of simulation results in real-time contributes to informed decision-making, allowing for a comprehensive understanding of potential impacts. Simultaneously, the typical tsunami warning protocol ensures seamless communication and data exchange with regional and international early warning systems. Timely dissemination of simulation results to relevant authorities is crucial for facilitating effective evacuation and emergency response efforts.

The PFT/FTRT workflow encompasses several key components, each contributing to the seamless and efficient operation of urgent computing in the context of tsunami simulations. These are listed in Table 18 and displayed in Figure 11.

Table 18. Principal parts of the Probabilistic Tsunami Forecast workflow.

#	Component	Description
1	Event data retriever	Source and tsunami data retrieval. (External)
2	Ensemble manager	Scenario sampler. Set scenarios to be simulated.
3	HySEA tsunami simulations	Parallel tsunami simulations on predefined topo-bathymetric grids and extraction of the information required for simulated time series to produce output files.
4	Simulations merging	Aggregation of simulations results into single netCDF file

5	Visualization	Quantification and visualization of the forecasts and potential definition of alert levels based on predefined rules defined by decision-makers.

In principle, any tsunami service provider (TCS) active at national level or within the tsunami warning systems coordinated by the Intergovernmental Coordination Group established by the Intergovernmental Oceanographic Commission (IOC) of UNESCO for the different areas of the world (European coasts are covered by the NEAMTWS, North-eastern Atlantic, the Mediterranean and connected seas Tsunami Warning System) may trigger the run of the workflow under the supervision of the person in charge (TCSs typically have a 24/7 service). The procedure may be automatic, that is based on predefined rules set on the potential for tsunami impact in selected target coasts for any given earthquake that occurs (e.g., based on earthquake magnitude and location). All TCS have all the required information for the initial input (seismic source) and the following updates (e.g. tsunami observations).

The person in charge would only have to use a unique command line for launching the workflow directly in the dedicated HPC environment. The command being pre-written with a adapted parameters and path dedicated to emergency cases as followed:

```
enqueue_compss \
  --num_nodes=500 \
  --exec_time=10 \
  $PWD/Code/launch_pycompss.py --seistype $seis_type --parameters_file $parfile
--event $event --data_path $data_path --run_path $workdir --templates_path
$templates_path --kagan_weight $kagan_weight --mare_weight $mare_weight --hours
$hours --group_sims=$group_sims --user_pois $user_pois --cfg from_template
```

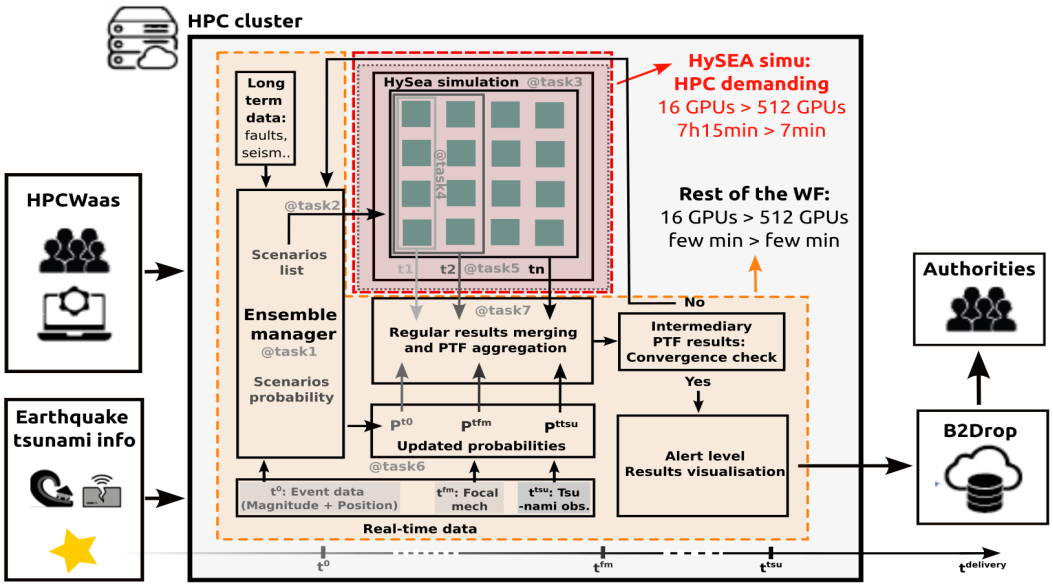

Figure 11. Overview of the eFlows4HPC Probabilistic Tsunami Forecast workflow.

Although the entire workflow benefits from HPC, the HySEA simulation segment requires the most resources. The full simulation's time of completion varies depending on several parameters which are: the number of simulations, the time of tsunami propagation and the domain of the simulations (size and resolution). For the Mediterranean version of the PTF, the domain is uniformly set to cover the entire Mediterranean Sea. The results are processed progressively and we can demonstrate that convergence is typically achieved for fewer than 500 scenarios. However, for large or particularly uncertain events, where a conservative approach is needed, we may need to demand that a higher number of scenarios (e.g. 1000 or more) are processed. The tsunami simulation needs to be adequate for the size of the domain time; for the Mediterranean cases, 8 hours is typically sufficient. The workflow performance times reported here are based upon 1000 simulations and 8h simulation time. (*Figure 12. Strong scaling results for the PTF workflow in the CTE-Power9 machine*) to bigger ressources (Figure 12). The required delivery time of the PTF to authorities in an emergency context should be in the order of 10 minutes, and it is followed by potential updates based on newly acquired information. This time is reached already with 512 GPUs. A single simulation requiring at least 7 minutes, the performance will reach a plateau with more GPUs. However the access to larger ressources (1024 GPUs) could allow higher grid resolution, more simulations.

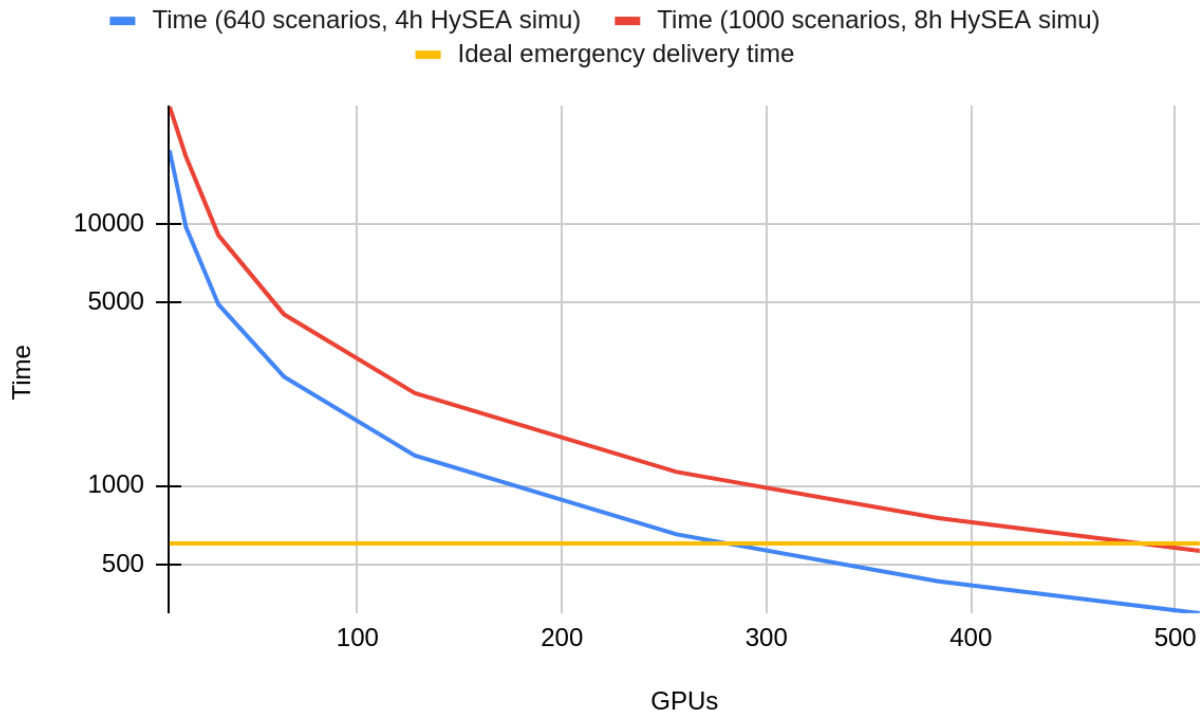

Figure 12. Time in seconds for completion of Tsunami-HySEA simulations for the number of scenarios and wave simulation duration indicated as a function of the number of GPUs employed. The ideal delivery time of 10 minutes is indicated with a straight line from which the minimum computational resources for the required load can be read. Numbers based on runs using the CTE-Power9 machine.

The details on the steps and outputs of a workflow run are described in Table 19. The advantage of the workflow in an emergency context is that the results could be immediately updated (Update of the ensemble step) as soon as new information on the event arrives without a need to do new

simulations. The new PTF results would be ready in a few seconds. If sufficient resources are available, all planned simulations could be run in parallel, and the forecast could be of immediate use. Otherwise, multiple cycles of simulations are required, in which convergence is monitored through an intermediate evaluation step. Notably, in this case, the convergence may be reached with a number of simulations that is smaller than the number originally planned, providing the evaluation sooner than expected.

Table 19. Time taken for different stages of the PTF workflow on CTE Power-9.

Kos-Bodrum test case					
Workflow run					
Environment	**Steps**	**Input** (Format, Size)	**Output** (Format, Size)	**Tested Resources: 16 GPUs**	**Ideal Resources: 512 GPUs**
User access : HPCWaas	User launch	line, command line		Depending on the queue (few seconds to hours)	
Workflow: HPC cluster	Ensemble manager	**ptf_main.config**: parameter file, number of scenarios: 1000, type of sampling LH (**txt, 200Ko**)	**Step1_scenario_list_BS.txt**: list of 1000 scenario's parameters (**txt, 200Ko**) **ptf_out.hdf5**: dictionary with scenarios parameters, probabilities and other dataset (**hdf5, 2Mo**)	27.7 minutes (reduced to few minutes without the jit library)	50 seconds
	Simulation preparation	**Step1_scenario_list_BS.txt**	**Step2_BS/BS_scenario_****/parfile.txt** : structure of 1000 folders with parameter file ready to run and output the HySea simulations (**txt, 100Mo**)	1.33 seconds	
	HySEA Simulations	**Step2_BS/BS_scenario_****/parfile.txt** : configuration of the simulations	**out_ts.nc**: file with the hmax of the simulations (**netcdf,1000*15Mo**)	**492 minutes (~8h10)**	7 minutes
	Hazard Curves Calculation (Intermediate and final results)	**out_ts.nc:** Hysea output **ptf_out.hdf5:** probabilities of the ensemble manager output	**out_ts_ptf.nc, Step2_BS_hmax.nc:** intermediate postprocessed files **(netcdf, 1Mo,3Mo)** **hazard_curves_RS_*.hdf5: (hdf5, 1.5Mo)** **ptf_out_*.hdf5: (hdf5,3.5Mo)**	- 2.72 seconds: post-processing - 7,249 seconds: hazard curves calculation + convergence and results visualization	< 1 minute < 1 minute

	Update of the ensemble	**tsunami_data.hdf5**: dictionary with the focal mechanism data and tide-gage data	**ptf_out.hdf5:** updated probabilities		
	Monitoring and convergence	**hazard_curves_RS_ *.hdf5** **out_ts.nc**	**conv_file.txt (txt, 100Ko)** **Conv_*_scenarios.png (png,250Ko)** ***_failed_scenario (txt, 100Ko)** ***_no_failed_scenario (txt, 100Ko)**		
	Results visualisation	**hazard_curves_RS_ *.hdf5** **ptf_out_*.hdf5**	***_scenarios_map_mean.png** ***_scenarios_histo_hazard_p*.png** ***_scenarios_map_p*.png**		
Results delivery: B2DROP	Results delivery	Result folder (<Go)	.tar compressed folder with 2 .txt file, 5 figures .png and 2 .hdf5 files (<Go)	5.2 seconds	seconds
Total time				**8h30 hours**	**8-10 minutes**

5.2 Earthquake

UCIS4EQ is a heterogeneous workflow that brings together the actions of different computational environments such as remote nodes or external servers, and HPC facilities. The schematic representation of the UCIS4EQ workflow is shown in Figure 13 where the purple box outlines the three building blocks currently deployed in HPC facilities.

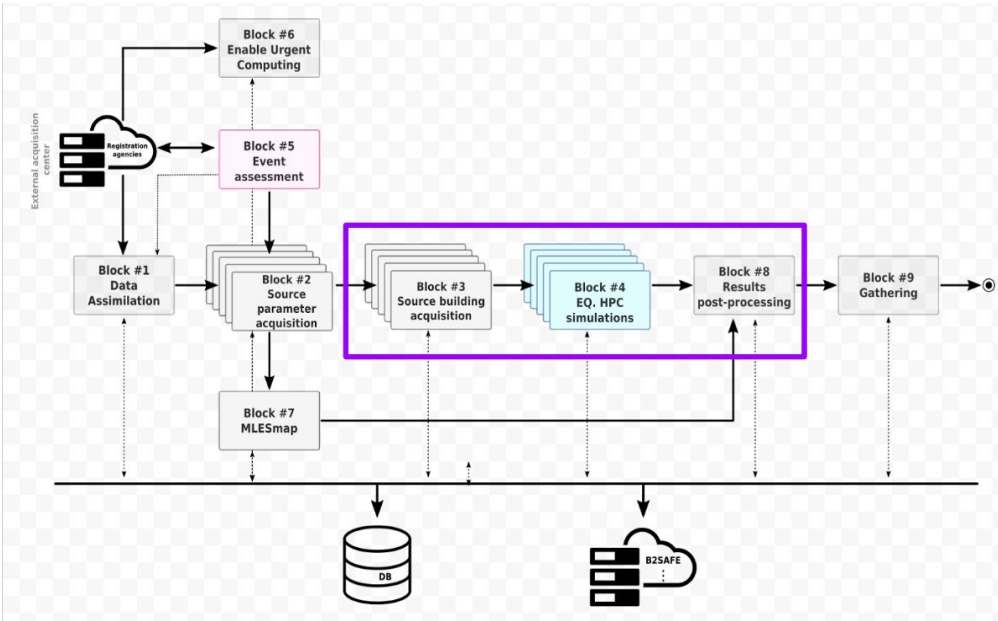

Figure 13. Schematic representation of UCIS4EQ. The purple box highlights the building blocks that run in the HPC environment.

We consider three main time intervals in an *urgent computing* (UC) operational context:

1. Elapsed time between the occurrence of a seismic event and registration by UCIS4EQ. Typically this time interval extends over a few minutes after the earthquake has occurred. International agencies in charge of collecting seismic event data in a standardized format[8] work around the clock and receive their information from local networks. During operations the *listener* service of UCIS4EQ must be also activated around the clock to pull information and select candidate-events for UC execution.
2. Queueing time in the computing infrastructure's execution queue. Without a UC protocol, a considerable amount of time is wasted waiting for job execution. This time mainly depends on the varying workload of HPC machines, on the type of queue selected to execute the job and on the queue manager performance optimization policy in place. In that regard a relevant objective of the eFlows4HPC project is to prepare recommendations to develop UC protocols, designed either to avoid or at least to minimize the queueing time in case of an emergency. HPC infrastructures Quality of Service (QoS, a.k.a. "job execution queue") policies are often opaque and complicate earnest queue performance optimization and job prioritisation attempts.
3. Execution time refers to the end-to-end execution of the UCIS4EQ workflow assuming no queueing times.

Following the time-fragmentation analysis above, this section focuses on the Execution time. The computation conducted on external nodes typically takes 3 to 5 minutes. A major portion of the execution time incurred by UCIS4EQ occurs in the HPC infrastructure as shown in Table 20. In particular, the *Salvus Run* service running the wave propagation simulations constitutes the bulk of computational requirements and is a key component in the Urgent Computing execution queue. The run times depicted in Table 20 are calculated for a single seismic source, and the number of nodes must be multiplied by the number of sources in the ensemble (i.e. the number of parallel simulations), as detailed later.

Table 20. Run time computed per each HPC-steps considering the configuration of Scenario 2 (Table 10).

HPC services	Nodes	Number of CPUs	Run time per job [minutes]
SlipGen	1	**1**	0.5
Salvus Prepare	1	**1**	0.7
Salvus Run	**16**	**768**	**14.6**
Salvus Post	1	**48**	8.6
Salvus Plot	1	**48**	0.2

[8] as mandated by the International Federation of Digital Seismograph Networks (FDSN)

Currently computational resources (i.e. the selection of the **queue** or **QoS** as well as the number and type(s) of nodes) used during HPC execution (Table 20) are pre-defined. They are specified before the execution of UCIS4EQ in the *Sites.json* file as shown in Figure 14. In an Urgent Computing Operational Framework *Sites.json* should be automatically configured depending on available and allocated resources.

```
"resources": {
  "SlipGenGP": {
    "wtime": 360,
    "nodes": 1,
    "tasks-per-node": 1,
    "qos": "debug",
    "environment": [
      "module load singularity"
    ]
  },
  "SalvusPrepare": {
    "wtime": 1800,
    "nodes": 1,
    "tasks-per-node": 1,
    "qos": "debug",
    "environment": [
      "module load ANACONDA/5.0.1",
      "source activate salvus"
    ]
  },
  "SalvusRun": {
    "wtime": 7200,
    "nodes": 10,
    "tasks-per-node": 48,
    "qos": "bsc_case",
    "environment":[
      "module load fabric",
      "export I_MPI_EXTRA_FILESYSTEM_LIST=gpfs",
      "export I_MPI_EXTRA_FILESYSTEM=on"
    ]
  },
  "SalvusPost": {
    "wtime": 1800,
    "nodes": 1,
    "tasks-per-node": 48,
    "qos": "debug",
    "environment": [
      "module load ANACONDA/5.0.1",
      "source activate salvus"
    ]
  },
  "SalvusPostSwarm": {
    "wtime": 1800,
    "nodes": 1,
    "tasks-per-node": 48,
    "qos": "debug",
    "environment": [
      "module load ANACONDA/5.0.1",
      "source activate salvus"
    ]
  },
  "SalvusPlots": {
    "wtime": 1800,
    "nodes": 1,
    "tasks-per-node": 48,
    "qos": "debug",
    "environment": [
      "module load ANACONDA/5.0.1",
      "source activate salvus"
```

Figure 14. Configuration file for the resources needed in the HPC-services provided in UCIS4EQ.

The computational cost of a seismic simulation is determined by multiple factors. Those must be taken into account for urgent computing to become an operational reality for earthquakes.

- **Simulation frequency.**

  Higher frequency simulations capture the dynamic and transient behaviour of structures during an earthquake better, as they yield more realistic results. Their predictions of how structures respond to seismic forces are more accurate, especially for complex or nonlinear behaviours. However, higher frequency simulations require greater computational resources: for a 3D medium, the computational cost of a wave propagation simulation scales with frequency to the fourth power. Doubling the simulation frequency, while maintaining all other simulation parameters unchanged, increases CPU time by a factor of 16 and memory requirements by 8. Therefore, the choice of simulation frequency is conditioned on the available computational resources available. In UC analysis, a trade-off emerges between the desired frequency and available computing resources. To note Salvus' binary compilation for GPU execution allows for higher frequency simulation than its binary equivalent for CPU execution while achieving a comparable or shorter time-to-solution.

- **Minimum and maximum wave velocity**

  The minimum and maximum speed of the waves depends on the geological properties of the subsurface. These values also influence the resources needed. If the minimum speed is reduced by half, the calculation time is 8 times longer. On the other hand, the maximum

speed is inversely related to the computational time. A maximum speed decrease by a factor of two entails a two-fold increase in calculation time.

- **Domain size**

  Increasing the size of the domain in an earthquake simulation can have various implications, both in terms of advantages and challenges.

  Improved Representation of the Affected Area: Increasing the domain size allows for a more realistic representation of the geographical extent affected by the earthquake. This is crucial for assessing seismic impact on large urban areas or extensive geographic regions.

  Consideration of Edge Effects: With a larger domain, edge effects and interactions at the periphery of the simulated area can be more accurately captured. This is important to avoid issues with seismic wave reflection at the boundaries of the domain.

  Inclusion of Relevant Geological Features: A larger domain may permit the inclusion of more complex geological features, such as additional seismic faults, variations in soil stiffness, and other elements that influence seismic behaviour.

  Higher Computational Demand: Increasing the domain size for a given frequency comes with higher computational demands. Larger models require more processing and memory resources, as well as longer absolute simulation times for waves to propagate through the entire domain. In particular, this parameter increases both the required computer resources and the time-to-solution. The computational cost increases linearly with the domain surface area for a fixed simulation length.

- **Ensemble size**

  In our application, we resort to using ensemble methods to account for uncertainties that arise as a result of a single instance of UCIS4EQ being activated by a potentially damaging earthquake ([9]; [13]). These ensemble methods enable the incorporation of different possible sources for the earthquake, based on variations of the geometry of the rupture fault and hypocentral location. Those possible variations in seismic source characteristics are the basis for an estimation of those uncertainties. Nevertheless, the size of this ensemble cannot be known a priori, because it depends mainly on the location of the earthquake and on the historical data catalogue available for that seismic region. Once the ensemble size is ascertained, UCIS4EQ launches parallel executions (one for each source), thereby multiplying the computer resources needed by the number of sources. Typically, the ensemble size has a few tens of sources. An increase in the number of sources considered in the ensemble method generally improves the uncertainty quantification in post-processing stages.

Each item above informs on a set of parameters that directly influence the time-to-solution as well as the computer resources needed for each UC simulation. Time to solution is thus a dynamic value that depends on many factors. For Urgent computing (UC) to fulfil its role, agreed upon objectives in terms of time-to-solution and result delivery to stakeholders must be met. Given those objectives in terms of time-to-solution and having identified the main parameters implicated in the seismic simulation, the number of nodes, memory, CPUs or GPUs can be determined.

We go on to assume that a proper post-event response time provided to National Seismological Services is ideally < 1hr. From this we may determine the computer resources needed to stage a single-source execution of UCIS4EQ (considering the simulation frequency, minimum and

maximum wave velocity, as well as domain size). We then multiply those resources by the number of seismic sources, determined on the fly, needed to compute an uncertainty.

UCIS4EQ has designed a building block named “Enable Urgent Computing” (BB #6, Fig. 5.3) with the main purpose of carrying out three main tasks:

1. Quantifying the priority to execute the earthquake under the urgent computing framework.
2. Set the computing resources needed for the simulation.
3. Activate the UC protocol and recommendations provided in this deliverable.

Since its early inception UCIS4EQ was designed as a workflow based on a microservice-architecture, where executions needed for the successful urgent processing of extreme seismic events are orchestrated. Beyond this Project, the continued development of the workflow functionalities is planned, whose main objective will remain to provide a timely response to stakeholders in case of emergencies.

5.3 LRMS configuration

All evaluation results presented in this document were obtained in the ‘standard’ way of acquiring resources on the system. The execution times (Walltime) were collected for different problem sizes (BSC, PSNC). The eflows4HPC Workflows were tested in the production HPC environment.

The evaluation on SLURM configuration for urgent computing mode was done in PSNC during the planned system maintenance. All users’ jobs were affected in the production environment and urgent code (SALVUS) was placed and run. The time expected for the initialisation of the job did not exceed 1 minute. The proposed policy encompasses the running job removal, instead of widely discussed - suspension. This is the outcome of the observation - every time on the system some jobs are placed with the requirement of a maximum amount of node local memory. There is no other fast memory available to shift the process image, that is why we decided to follow the kill approach.

Both systems located at BSC as well as in PSNC are managed by queueing system SLURM. This is the most popular resource management system in the current HPC data centres.

PSNC resources:

> EAGLE: 63360 cores (Intel Xeon Platinium 8268), 78 GPU cards (Nvidia V100), 1320 nodes, RAM 192/384 GB

BSC resources:

> Marenostrum 4: 165888 cores (Intel Xeon Platinum), 208 GPU cards (Nvidia V100), 3456 nodes, RAM 96/384 GB

SLURM efficiently manages and optimises the allocation of computing resources. Urgent computing applications can benefit from SLURM's ability to allocate resources dynamically, ensuring that the required resources are available on time. It supports job dependencies, allowing users to define relationships between different jobs. In general, this feature is beneficial for urgent computing workflows where certain tasks depend on the completion of others. The deployed in the project PyCOMPSs workflows are run as a single job with all its internal dependencies. It simplifies the workload management on the system. SLURM is responsible for collecting the

resources and yielding the exit codes of the workflow execution, handing over the steering of the flow to the requestor.

The example of SLURM configuration including urgent computing support is presented in the **Appendix 1**.

6 Acronyms and Abbreviations

- CA – Consortium Agreement
- CC - Cyclomatic Complexity
- CPU - Central Processing Unit
- D – deliverable
- DoA – Description of Action (Annex 1 of the Grant Agreement)
- EB – Executive Board
- EC – European Commission
- GA – General Assembly / Grant Agreement
- GPU - Graphics Processing Unit
- HPC – High Performance Computing
- IPR – Intellectual Property Right
- KPI – Key Performance Indicator
- M – Month
- MS – Milestones
- LLoC - Logical Lines of Code
- LRMS - Local Resources Management System
- PM – Person month / Project manager
- PTF - Probabilistic Tsunami Forecast
- SLURM - popular queueing system
- TRL – Technology Readiness Level
- TCS - Thematic Core Service
- UC – Urgent Computing
- UCIS4EQ – Urgent Computing Integrated Services for Earthquakes
- WF – Workflow
- WM – Workflow Manager
- WP – Work Package
- WPL – Work Package Leader

7 References

1. SLURM Gang Scheduling, https://slurm.schedmd.com/gang_scheduling.html
2. SLURM Preemption, https://slurm.schedmd.com/preempt.html
3. Manolis Marazakis, et al., HPC for Urgent Decision-Making, White Paper, etp4hpc.eu, 15/02/2022 https://doi.org/0.5281/zenodo.6107362
4. Gordon Gibb, Rupert Nash, Nick Brown, Bianca Prodan, The Technologies Required for Fusing HPC and Real-Time Data to Support Urgent Computing, arXiv:2010.01543v1 [cs.DC] 4 Oct 2020
5. K.K. Yoshimotoa, et. All., Implementations of Urgent Computing on Production HPC Systems, Procedia Computer Science, Volume 9, 2012, Pages 1687-1693
6. Urgent Computing Policy recommendation, eflows4Hpc White paper, 2024, https://eflows4hpc.eu/publications/
7. Bradner, S. O. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119. Request for Comments. https://rfc-editor.org/rfc/rfc2119.txt, 1997.
8. McCabe, T. J. 1976. A Complexity Measure., IEEE Transactions on Software Engineering, SE-2 (4), pp. 308-320 https://doi.org/10.1109/TSE.1976.233837
9. Monterrubio-Velasco, M., Carrasco-Jimenez, J. C., Rojas, O., Rodriguez, J. E., Fichtner, A., & Puente, J. D. L. (2022). A statistical approach towards fast estimates of moderate-to-large earthquake focal mechanisms. Frontiers in Earth Science, 10, 743860.
10. Monterrubio-Velasco, M., Pienlowska M., Bhihe C., Blanco-Prieto R., Iglesias A., de la Puente J. UCIS4EQ applied to the M7.1 2017 earthquake in Puebla (México). *ESS Open Archive* . February 02, 2024a. DOI: 10.22541/essoar.170689118.88369036/v1
11. Monterrubio-Velasco, M., Blanco-Prieto, R., Callaghan, S., de la Puente J. Machine Learning based Estimator for Ground Shaking Maps in South Iceland Seismic Zone (SISZ). ESS Open Archive . February 02, 2024b. DOI: 10.22541/essoar.170689203.31217011/v1
12. M. Agung, Y. Watanabe, H. Weber, R. Egawa and H. Takizawa, "Preemptive Parallel Job Scheduling for Heterogeneous Systems Supporting Urgent Computing," in *IEEE Access*, vol. 9, pp. 17557-17571, 2021, doi: 10.1109/ACCESS.2021.3053162.
13. Cordrie, L., Selva, J., Bernardi, F., and Tonini, R.: Using available and incoming data for reducing and updating seismic source ensembles for probabilistic tsunami forecasting (PTF) in early-warning and urgent computing, EGU General Assembly 2023, Vienna, Austria, 24–28 Apr 2023, EGU23-12363, https://doi.org/10.5194/egusphere-egu23-12363, 2023.

8 List of figures and tables

9 Appendix A

SLURM configuration file for EAGLE@PSNC

```
#Note! Some parameters may have been hidden due to security reason
AuthAltTypes=auth/jwt
AuthAltParameters=jwt_key=****
SlurmctldParameters=enable_configless
ClusterName=eagle
ControlMachine=****
ControlAddr=****
StateSaveLocation=/var/lib/slurm-llnl/slurmctld
SlurmUser=slurm
SlurmdUser=root
SlurmdPidFile=/var/run/slurmd.pid
SlurmdPort=****
SlurmdSpoolDir=/var/lib/slurm-llnl/slurmd
SlurmdTimeout=1200
SlurmctldPidFile=/var/run/slurm-llnl/slurmctld.pid
SlurmctldPort=****-****
SlurmctldTimeout=60
AuthType=auth/munge
AuthInfo="****"
CryptoType=crypto/munge
#Licenses=app:nn,...
MailProg=/usr/bin/mail-slurm.py
MaxArraySize=65533
MaxJobCount=45000
MaxStepCount=45000
MaxTasksPerNode=48
MinJobAge=300
AllowSpecResourcesUsage=0
BatchStartTimeout=10
GroupUpdateForce=0
```

GroupUpdateTime=600

CompleteWait=0

DisableRootJobs=YES

EioTimeout=60

EnforcePartLimits=NO

FirstJobId=* #hidden

MaxJobId=* #hidden

InactiveLimit=0

JobFileAppend=1

JobRequeue=1

KillOnBadExit=1

KillWait=30

LaunchType=launch/slurm

OverTimeLimit=0

PrivateData=cloud

Proctracktype=proctrack/cgroup

PropagatePrioProcess=0

PropagateResourceLimits=NONE

ResvOverRun=0

ReturnToService=2

SwitchType=switch/none

TmpFs=/tmp

TrackWCKey=NO

TreeWidth=40

VSizeFactor=0

WaitTime=0

Epilog=/etc/cluster/scripts/epilog.sh

Prolog=/etc/cluster/scripts/prolog.sh

EpilogMsgTime=2000

PrologFlags=x11

X11Parameters=home_xauthority

PluginDir=/usr/lib/x86_64-linux-gnu/slurm

PlugStackConfig=/etc/slurm/plugstack.conf

CoreSpecPlugin=core_spec/none

ExtSensorsType=ext_sensors/none

RoutePlugin=route/default

TaskPlugin=task/cgroup,task/affinity

TopologyPlugin=topology/tree

SchedulerType=sched/backfill

SchedulerParameters=bf_continue,bf_interval=600,bf_max_job_part=100,bf_max_job_start=300,bf_max_job_test=300,bf_max_job_user=20,bf_resolution=120,max_rpc_cnt=150

SchedulerTimeSlice=30

FairShareDampeningFactor=1

SelectType=select/cons_res

SelectTypeParameters=CR_Core_Memory

PreemptType=preempt/partition_prio

PriorityType=priority/multifactor

PriorityDecayHalfLife=14-0

PriorityCalcPeriod=5

PriorityFavorSmall=NO

PriorityFlags=FAIR_TREE,SMALL_RELATIVE_TO_TIME,MAX_TRES

PriorityMaxAge=7-0

PriorityUsageResetPeriod=NONE

PriorityWeightAge=1000

PriorityWeightFairshare=10000

PriorityWeightJobSize=10000000

PriorityWeightPartition=1000

PriorityWeightQOS=10000

AccountingStorageEnforce=associations,limits

AccountingStorageHost=eagle-slurm-2

AccountingStoragePass=/var/run/munge/munge.socket.2

AccountingStoragePort=****

AccountingStorageTRES=gres/gpu,license/abaqus,cpu,mem,node

AccountingStorageType=accounting_storage/slurmdbd

AccountingStoreFlags=job_comment,job_script

JobCompType=jobcomp/elasticsearch

JobCompLoc=http://*.*.*.*:*/eagle_slurm/_doc

```
JobCompParams=connect_timeout=5
JobAcctGatherType=jobacct_gather/cgroup
JobAcctGatherFrequency="task=30,energy=60,network=60,filesystem=60"
AcctGatherProfileType=acct_gather_profile/influxdb
AcctGatherNodeFreq=120
AcctGatherEnergyType=acct_gather_energy/ipmi
AcctGatherInfinibandType=acct_gather_infiniband/ofed
AcctGatherFilesystemType=acct_gather_filesystem/lustre
AcctGatherInterconnectType=acct_gather_interconnect/ofed
HealthCheckInterval=120
HealthCheckNodeState=CYCLE
HealthCheckProgram=/usr/sbin/nhc
SlurmctldDebug=3
SlurmctldLogFile=/var/log/slurm-llnl/slurmctld.log
SlurmdDebug=3
SlurmdLogFile=/var/log/slurm-llnl/slurmd.log
SlurmSchedLogLevel=2
SlurmSchedLogFile=/var/log/slurm-llnl/slurmsched.log
CpuFreqDef=Performance
RebootProgram=/etc/cluster/scripts/reboot_program.sh
SuspendProgram=/etc/cluster/scripts/suspend_program.sh
ResumeProgram=/etc/cluster/scripts/resume_program.sh
SuspendTimeout=60
ResumeTimeout=1800
ResumeRate=100
SuspendRate=200
SuspendTime=1800
NodeName=e[1088-2088,2137-2152] CPUs=48 Sockets=2 CoresPerSocket=24 ThreadsPerCore=1 State=UNKNOWN RealMemory=191937 Feature=intel,cascade,edr,huawei,192GB Weight=1 MemSpecLimit=2048
NodeName=e[2089-2136,2153-2408] CPUs=48 Sockets=2 CoresPerSocket=24 ThreadsPerCore=1 State=UNKNOWN RealMemory=383887 Feature=intel,cascade,edr,huawei,384GB Weight=200 MemSpecLimit=2048
GresTypes=gpu
```

NodeName=gpu01 CPUs=20 Sockets=2 CoresPerSocket=10 ThreadsPerCore=1 State=UNKNOWN RealMemory=191837 Feature=gpu Gres=gpu:tesla:2 MemSpecLimit=2048

NodeName=gpu02 CPUs=20 Sockets=2 CoresPerSocket=10 ThreadsPerCore=1 State=UNKNOWN RealMemory=757073 Feature=gpu Gres=gpu:tesla:2 MemSpecLimit=2048

NodeName=gpu03 CPUs=20 Sockets=2 CoresPerSocket=10 ThreadsPerCore=1 State=UNKNOWN RealMemory=191837 Feature=gpu Gres=gpu:tesla:2 MemSpecLimit=2048

NodeName=gpu[04-12] CPUs=32 Sockets=2 CoresPerSocket=16 ThreadsPerCore=1 State=UNKNOWN RealMemory=383887 Feature=gpu Gres=gpu:tesla:8 MemSpecLimit=2048

PartitionName=altair Nodes=e[1088-2408] AllocNodes=e[0001-2408],AllowGroups=admins,staff,users DefMemPerCPU=2048 DefaultTime=24:00:00 MaxTime=168:00:00 Priority=10000 State=UP Hidden=NO Default=NO Shared=YES **PreemptMode=requeue**

PartitionName=altair-gpu Nodes=gpu[04-12] AllocNodes=eagle AllowGroups=admins,staff,users DefMemPerCPU=2048 DefaultTime=24:00:00 MaxTime=168:00:00 Priority=10000 State=DOWN Hidden=NO Default=NO Shared=YES

PartitionName=interactive Nodes=e[2000-2400] AllocNodes=e[2000-2400] AllowGroups=admins,staff,users DefMemPerCPU=2048 DefaultTime=01:00:00 MaxTime=10:00:00 Priority=10000 State=UP Hidden=NO Default=NO Shared=YES TRESBillingWeights="CPU=1.0,Mem=0.5G"

PartitionName=standard Nodes=e[1089-2408] AllocNodes=e[1089-2408] AllowGroups=admins,staff,users DefMemPerCPU=2048 DefaultTime=24:00:00 MaxTime=168:00:00 Priority=1 State=UP Hidden=NO Default=YES Shared=YES **PreemptMode=requeue**

PartitionName=**urgent** Nodes=e[1089-2408] AllocNodes=e[1089-2408] AllowGroups=admins,**urgentusers** DefMemPerCPU=2048 DefaultTime=24:00:00 MaxTime=168:00:00 Priority=30000 State=UP Hidden=NO Default=NO Shared=YES **PreemptMode=off**

PartitionName=long Nodes=e[1089-2408] AllocNodes=e[1089-2408] AllowGroups=admins,staff,users DefMemPerCPU=2048 DefaultTime=24:00:00 MaxTime=504:00:00 Priority=20000 State=UP Hidden=NO Default=NO Shared=YES **PreemptMode=requeue**

PartitionName=fast Nodes=e[1089-2408] AllocNodes=e[1089-2408] AllowGroups=admins,staff,users DefMemPerCPU=2048 DefaultTime=00:15:00 MaxTime=01:00:00 Priority=1000 State=UP Hidden=NO Default=NO Shared=YES **PreemptMode=requeue**

PartitionName=tesla Nodes=gpu[01-12] AllocNodes=e[1088-2408] AllowGroups=admins,staff,users DefMemPerCPU=2048 DefaultTime=24:00:00 MaxTime=168:00:00 Priority=1 State=UP Hidden=NO Default=NO Shared=YES

10 Appendix B

#Original PTF bash script equivalent to the PTF Workflow.

```
#!/bin/bash
#############################
# SETTINGS
#
# eventID: it controls the target event
#       existing events in IO:  2020_1030_samos, 2018_1025_zante
#
# domain: it controls the target area, target points, and relative settings
#       existing domains in IO:  med-tsumaps, pacific-cheese
#
# tsu_sim: it controls the managmenent of simulations. We have 3 options:
#       tsu_sim=torun: run simulations through t-hysea;
#       tsu_sim=completed:simulations results aready in IO folder;
#       tsu_sim=precomputed: simulations from hazard (only domain=med-tsumaps)
#
#############################
eventID=2003_0521_boumardes
domain=med-tsumaps
tsu_sim=precomputed

step1YN=true
step2YN=false
step3YN=false
step4YN=false
step5YN=true

#############################
# CONFIGURATION
#############################
sigma=4 #controlling the extension of exploration of uncertainty, see Selva et al. (2021), Nat. Comm.
showmat=false   # true: show matlab; false: write matlab diary
if [ $tsu_sim == 'torun' ]; then
## SETTINGS FOR STEP 2 - EXPLICIT SIMULATIONS - ##
        sim_domain=regional                     #simulation domain extension: local; regional; global
        if [ $sim_domain == 'local' ]; then
        bathyfile=$sim_domain\_domain_$eventID\.grd
        depthfile=$sim_domain\_domain_$eventID\_POIs_depth.dat
        else
        bathyfile=$sim_domain\_domain.grd
        depthfile=$sim_domain\_domain_POIs_depth.dat
        fi
        propagation=8                           #hours of tsunami propagation
        host=M100                               #Marconi100 @CINECA => Scheduler: SLURM; MC HySEA 3.8.1
        username=mvolpe00
fi
alert_lev=true # true: compute alert levels; false: no alert levels
#modules
localserver=$(hostname)
echo $localserver
if [ $localserver == 'argo' ]; then
        module purge
        module load matlab
        module load gcc/8.1.0 hdf5/1.8.18/gcc-8.1.0 netcdf/4.6.1/gcc-8.1.0
        module load python/3.4.0
        module load GMT/5.4.3
        mainFolder=$(pwd)
elif [ $localserver == 'auriga' ]; then
        module purge
        module load matlab
        module load GMT/5.4.2
        module load python/3.4.0
        mainFolder=$(pwd)
else
        mainFolder=$(pwd)
fi

workdir=$mainFolder/IO/$domain\_$eventID
echo $workdir
if [ ! -d $workdir ]; then
```

```
        mkdir $workdir
        mkdir $workdir/step3_output
        mkdir $workdir/step1_output
        chmod 775 $workdir
else
        echo 'Output file already exists: results will be overwritten'
fi

if [ ! -d $workdir/step3_output ]; then
        mkdir $workdir/step3_output
        chmod 775 $workdir/step3_output
else
        echo 'Output file already exists: results will be overwritten'
fi

if [ ! -d $workdir/step1_output ]; then
        mkdir $workdir/step1_output
        chmod 775 $workdir/step1_output
else
        echo 'Output file already exists: results will be overwritten'
fi

cp $mainFolder/IO/earlyEst/$eventID\_stat.json $workdir

##############################
# RUN SCRIPTS
##############################
echo 'Event:     '$eventID
echo 'Domain:    '$domain
echo 'Run type: '$tsu_sim
echo 'Workdir:  '$workdir

#########
# STEP 1
#########
echo '============================'
echo '========== STEP 1 =========='
echo '============================'
if $step1YN; then
  echo 'run_step1.py --cfg ../cfg/ptf_main.config --event ../IO/earlyEst/'$eventID'_stat.json'
  cd $mainFolder/Step1_EnsembleDef_python
  python ./run_step1.py --cfg ../cfg/ptf_main.config --event ../IO/earlyEst/$eventID\_stat.json
  mv ptf_localOutput/* $workdir/step1_output/
  mv $workdir/step1_output/ptf_out.hdf5 $mainFolder/Step3_HazardAggregation_python/ptf_localOutput/
else
   echo 'Not exectuted'
fi

#########
# STEP 2
#########
echo '============================'
echo '========== STEP 2 =========='
echo '============================'
if $step2YN; then

  if [ $tsu_sim == 'torun' ]
  then
        echo 'Creating setup for simulations'
        cd $mainFolder/Step2_RunSimulations
        inpfileBS=$workdir/step1_output/Step1_scenario_list_BS.txt
        inpfilePS=$workdir/step1_output/Step1_scenario_list_PS.txt
        npois=$(wc $mainFolder/IO/$domain/POIs.txt | awk '{print $1}')
        sed "1i$npois" < $mainFolder/IO/$domain/POIs.txt > $workdir/Step2_ts.dat
        ln -sf $mainFolder/IO/$domain/POIs.txt $workdir/Step2_POIs.txt
        ln -sf $mainFolder/IO/$domain/bathy_grids/$depthfile $workdir/Step2_$depthfile
        ln -sf $mainFolder/IO/$domain/bathy_grids/$bathyfile $workdir/Step2_$bathyfile
        echo '--------------'
        echo 'Preparing submission scripts for ' $host
        cp Step2_launch_simul_$host\.sh $workdir
        if [ -f $inpfileBS ]; then
        cp Step2_simul_BS.sh $workdir
        cp Step2_parfile_tmp.txt $workdir
        fi
```

```
        if [ -f $inpfilePS ]; then
        cp Step2_simul_PS.sh $workdir
        cp Step2_parfile_TRI_tmp.txt $workdir
        fi
        cp Step2_extract_ts.py $workdir
        cp Step2_create_ts_input_for_ptf.py $workdir

        if [ $host == 'M100' ]; then
        cp Step2_requirements_$host\.source $workdir
        remote_path=/m100_scratch/userexternal/$username/ChEESE_PD8/$domain\_$eventID         #the   folder
ChEESE_PD8 is supposed to be there
        code_path=/m100/home/userexternal/csanchez/HySEA_v3.8.1_MC
        #account=Pra21_5386
        account=Pra20_5169                      #Account name of the assigned resources  (TO BE DEFINED:
THIS IS EXIPIRED)
        partition=m100_usr_prod                 #Partition for the resource allocation
        quality=normal                          #Quality of service for the job (max = 16 nodes)
        nnodes=1                                #1 node / job (4 GPUs)
        nsims=4                                 #4 simulations / job
        walltime=00:30:00                       #maximum excution time for each job (hh:mm:ss)
        echo '#!/bin/bash -l' > $workdir/Step2_run_tmp.sh
        echo ' ' >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --job-name=TSUNAMI_SIM_XX_ZZZ' >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --account='$account >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --partition='$partition >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --output=logfiles_XX/log_XX_ZZZ_%j.out' >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --qos='$quality >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --nodes='$nnodes >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --ntasks='$nsims >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --ntasks-per-node=4' >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --cpus-per-task=4' >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --gres=gpu:4' >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --exclusive' >> $workdir/Step2_run_tmp.sh
        echo '#SBATCH --time='$walltime >> $workdir/Step2_run_tmp.sh
        echo ' ' >> $workdir/Step2_run_tmp.sh
        #echo 'module load python/3.8.2' >> $workdir/Step2_run_tmp.sh
        echo 'source Step2_requirements_'$host.source >> $workdir/Step2_run_tmp.sh
        echo 'cd '$remote_path >> $workdir/Step2_run_tmp.sh
        echo 'mpirun -np NUMPROC ./TsunamiHySEA INPUTFILE' >> $workdir/Step2_run_tmp.sh
        echo 'while read line; do' >> $workdir/Step2_run_tmp.sh
        echo '  simdir=$(dirname $line)' >> $workdir/Step2_run_tmp.sh
        echo '  python Step2_extract_ts.py --nc $simdir/out_ts.nc --depth_file Step2_'$depthfile '--ptf_file
$simdir/out_ts_ptf.nc' >> $workdir/Step2_run_tmp.sh
        echo 'done < INPUTFILE' >> $workdir/Step2_run_tmp.sh
        echo 'rm -f INPUTFILE' >> $workdir/Step2_run_tmp.sh
        ssh -i ~/.ssh/id_rsaM100 $username@login.m100.cineca.it mkdir $remote_path
        ssh  -i  ~/.ssh/id_rsaM100  $username@login.m100.cineca.it  ln  -sf  $code_path/bin/TsunamiHySEA
$remote_path
        ssh -i ~/.ssh/id_rsaM100 $username@login.m100.cineca.it ln -sf $code_path/bin_lb/get_load_balancing
$remote_path
        echo 'Trasferring data to' $host
        if [ -f $inpfileBS ]; then
        scp -i ~/.ssh/id_rsaM100 $inpfileBS $username@login.m100.cineca.it:$remote_path
        fi
        if [ -f $inpfilePS ]; then
        scp -i ~/.ssh/id_rsaM100 $inpfilePS $username@login.m100.cineca.it:$remote_path
        if      ssh      -i      ~/.ssh/id_rsaM100      $username@login.m100.cineca.it      [      !      -d
/m100_scratch/userexternal/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1 ]; then
                scp     -i     ~/.ssh/id_rsaM100     $mainFolder/IO/$domain/INIT_COND_PS_TSUMAPS1.1.tar.bz2
$username@login.m100.cineca.it:/m100_scratch/userexternal/$username/ChEESE_PD8
                ssh        -i        ~/.ssh/id_rsaM100        $username@login.m100.cineca.it        bunzip2
/m100_scratch/userexternal/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1.tar.bz2
                ssh       -i       ~/.ssh/id_rsaM100       $username@login.m100.cineca.it       tar       -xvf
/m100_scratch/userexternal/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1.tar       #this takes some time (but
needed just once)
        fi
        ssh         -i         ~/.ssh/id_rsaM100         $username@login.m100.cineca.it         ln         -sf
/m100_scratch/userexternal/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1 $remote_path
        fi
        scp -i ~/.ssh/id_rsaM100 $workdir/Step2_$bathyfile $username@login.m100.cineca.it:$remote_path
        scp -i ~/.ssh/id_rsaM100 $workdir/Step2_$depthfile $username@login.m100.cineca.it:$remote_path
        scp -i ~/.ssh/id_rsaM100 $workdir/Step2_ts.dat $username@login.m100.cineca.it:$remote_path
        scp -i ~/.ssh/id_rsaM100 $workdir/Step2_parfile*  $username@login.m100.cineca.it:$remote_path
```

```
        scp            -i            ~/.ssh/id_rsaM100            $workdir/Step2_requirements_$host\.source
$username@login.m100.cineca.it:$remote_path
        scp -i ~/.ssh/id_rsaM100 $workdir/*.sh  $username@login.m100.cineca.it:$remote_path
        scp -i ~/.ssh/id_rsaM100 $workdir/*.py  $username@login.m100.cineca.it:$remote_path

        elif [ $host == 'auriga' ]; then
        echo 'todo'
        elif [ $host == 'cat-scenari2' ]; then
        echo 'mpirun -np 2 TsunamiHySEA_2GPU_grd parfile' > $workdir/Step2_run_tmp.sh
        echo 'python Step2_extract_ts.py --nc simdir/out_ts.nc --depth_file' Step2_$depthfile '--ptf_file
simdir/out_ts_ptf.nc' >> $workdir/Step2_run_tmp.sh
        remote_path=/data/users/$username/ChEESE_PD8/$domain\_$eventID   #the folder ChEESE_PD8 is supposed
to be there
        ssh $username@cat-scenari2 mkdir $remote_path
        echo 'Trasferring data to' $host
        if [ -f $inpfileBS ]; then
        scp $inpfileBS $username@cat-scenari2:$remote_path
        fi
        if [ -f $inpfilePS ]; then
        scp $inpfilePS $username@cat-scenari2:$remote_path
        if ssh $username@cat-scenari2 [ ! -d /data/users/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1 ]; then
                scp          $mainFolder/IO/$domain/INIT_COND_PS_TSUMAPS1.1.tar.bz2          $username@cat-
scenari2:/data/users/$username/ChEESE_PD8
                ssh                          $username@cat-scenari2                          bunzip2
/data/users/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1.tar.bz2
                ssh                  $username@cat-scenari2                  tar                  -xvf
/data/users/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1.tar     #this takes some time (but needed just once)
        fi
        ssh   $username@cat-scenari2   ln   -sf   /data/users/$username/ChEESE_PD8/INIT_COND_PS_TSUMAPS1.1
$remote_path
        fi
        scp $workdir/Step2_$bathyfile $username@cat-scenari2:$remote_path
        scp $workdir/Step2_$depthfile $username@cat-scenari2:$remote_path
        scp $workdir/Step2_ts.dat $username@cat-scenari2:$remote_path
        scp $workdir/Step2_parfile* $username@cat-scenari2:$remote_path
        scp $workdir/*.sh  $username@cat-scenari2:$remote_path
        scp $workdir/*.py  $username@cat-scenari2:$remote_path

        #elif [ $host == 'XXXX' ]; then
        fi
        echo '...run T-HySea'
        echo '--------------'
   #####ON THE REMOTE HPC SERVER####
        cd $remote_path
        if [ -f $inpfileBS ]; then
        nohup ./Step2_launch_simul_$host\.sh BS Step2_$bathyfile $propagation > nohup_BS.out &
        fi
        if [ -f $inpfilePS ]; then
        nohup ./Step2_launch_simul_$host\.sh PS Step2_$bathyfile $propagation > nohup_PS.out &
        fi
        python Step2_create_ts_input_for_ptf.py --depth_file Step2_$depthfile # Once ALL the simulations have
been completed  (check squeue?)
   ####BACK TO LOCAL SERVER####
        if [ -f $inpfileBS ]; then
        if [ $host == 'M100' ]; then
        scp -i ~/.ssh/id_rsaM100 $username@login.m100.cineca.it:$remote_path/Step2_BS_hmax.nc $workdir
        elif [ $host == 'cat-scenari2' ]; then
        scp $username@cat-scenari2:$remote_path/Step2_BS_hmax.nc $workdir
        fi
        fi
        if [ -f $inpfilePS ]; then
        if [ $host == 'M100' ]; then
        scp -i ~/.ssh/id_rsaM100 $username@login.m100.cineca.it:$remote_path/Step2_PS_hmax.nc $workdir
        elif [ $host == 'cat-scenari2' ]; then
        scp $username@cat-scenari2:$remote_path/Step2_PS_hmax.nc $workdir
        fi
        fi

  else
        echo '...Step 2 is not necessary'
  fi
else
   echo 'Not exectuted'
fi
```

```
#########
# STEP 3
#########
echo '============================'
echo '========== STEP 3 ==========='
echo '============================'
if $step3YN; then
  cd $mainFolder/Step3_HazardAggregation_python
  echo '-------STEP 3-------'
  python ./run_step3.py --cfg ../cfg/ptf_main.config --event ../IO/earlyEst/$eventID\_stat.json
  mv $mainFolder/Step3_HazardAggregation_python/ptf_localOutput/* $workdir/step3_output/
else
   echo 'Not exectuted'
fi

#########
# STEP 4
#########
echo '============================'
echo '========== STEP 4 ==========='
echo '============================'
if $step4YN; then
  if $alert_lev; then
        cd $mainFolder/Step4_AlertLevels
        echo '-------STEP 4-------'
        echo 'matlab -nodisplay -nosplash -nodesktop -r "Step4_run('$eventID','$domain'); exit;"'
        if $showmat; then
        matlab -nodisplay -nosplash -nodesktop -r "Step4_run('$eventID','$domain'); exit;"
        else
        matdiary=$workdir/Step4_MatlabDiary.txt
        echo "matlab output in $matdiary"
        matlab -nodisplay -nosplash -nodesktop -r "Step4_run('$eventID','$domain'); exit;"   > $matdiary
        fi

  else
        echo '...Step 4 is not necessary'
  fi
else
   echo 'Not exectuted'
fi

#########
# STEP 5
#########
echo '============================'
echo '========== STEP 5 ==========='
echo '============================'
if $step5YN; then
  if ! $step3YN; then
        cp $mainFolder/Step3_HazardAggregation_python/ptf_localOutput/ptf_out* $workdir/step3_output/
  fi
  cd $mainFolder/Step5_Visualization_python
  echo '-------STEP 5-------'
  python3  ./run_step5.py  --cfg  ../cfg/ptf_main.config  --event  ../IO/earlyEst/$eventID\_stat.json  --
save_main_path $workdir
else
   echo 'Not exectuted'
fi
```

11 Appendix C

PyCOMPSs version of the PTF Workflow

```
###################### PYCOMPSS TASKS #########################
@constraint(computing_units=big_task_cu)
@task(config_file=FILE_IN, returns=1)
def step1_func(args, config_file, seistype, sim_files_step1):
        args.cfg=config_file
        run_step1_init(args,sim_files_step1)
        return sim_files_step1 + "/Step1_scenario_list_"+seistype+".txt"

@binary(binary=config_bin)
@task(config_template=FILE_IN, config_file=FILE_OUT, par_file=FILE_IN)
def  build_config(config_template,  config_file,  data_dir,  files_step2,  par_file,  kag,  tsu,  event_id,
user_pois):
        pass

@constraint(processors=[{'processorType':'CPU', 'computingUnits':'1'},
                        {'processorType':'GPU', 'computingUnits':'1'}])
@mpi(binary=tsunamiHySEA_bin,       args="{{file_in}}",       runner="mpirun",       processes=gpus_per_exec,
processes_per_node=gpus_per_node,working_dir="{{wdir}}")
@task(file_in=FILE_IN,returns=1)
def mpi_func(file_in, wdir):
        pass

@constraint(computing_units=big_task_cu)
@binary(binary=simulBS_bin, working_dir="{{wdir}}")
@task(grid=FILE_IN, pois_ts_file=FILE_IN, sim_template_file=FILE_IN, sim_files_step2=FILE_OUT)
def  build_structure(seistype,  grid,  hours,  group,  sim_files_step2,  load_balancing,  pois_ts_file,
sim_template_file, sim_events_files, wdir):
        pass

#@task(log_file_sim=FILE_OUT)
@constraint(computing_units=small_task_cu)
@task(depth_file=FILE_IN, returns=1)
def extract_ts(out_ts,depth_file,ptf_file,simwdir,centinel):
        step2_extract_ts(out_ts,depth_file,ptf_file,simwdir)

@constraint(computing_units=big_task_cu)
@task(returns=1)
def create_ptf_input(ptf_files,out_path,depth_file,log_file):
        step2_create_ptf_input(ptf_files,out_path,depth_file,log_file)

@constraint(computing_units=small_task_cu)
@task(ptf_files=COMMUTATIVE, config_file=FILE_IN, depth_file=FILE_IN)
def  append_and_evaluate(ptf_files,  ptf_file,  args,  config_file,  sim_files_step1,  out_step2_path,
out_update_path, out_final, depth_file, log_file, sim_pois_ts, num_sims, kag, tsu, result_ext, conv_file,
fig_path):
        args.cfg = config_file
        ptf_files.append(ptf_file)
        if (num_sims != 0) and (len(ptf_files) % num_sims == 0):
        fail=step2_create_ptf_input(ptf_files, out_step2_path, depth_file, log_file)
        if kag>0:
        run_step_kagan(args,sim_files_step1,out_update_path)
        sim_files_input=out_update_path
        elif tsu>0:
        run_step_mare(args, sim_files_step1, out_update_path, sim_pois_ts, ptf_files)
        sim_files_input=out_update_path
        else:
        sim_files_input=sim_files_step1
        run_step3_init(args, sim_files_input, out_final, sim_pois_ts, ptf_files, fail, conv_file, fig_path)

@binary(binary="tar", args="zcvf {{outfile}} {{folder}}")
@task(outfile=FILE_OUT)
def compress(folder, outfile, ptf_files):
        pass
```

```
######################## MAIN SCRIPT ############################
if __name__ == '__main__':
   # reading arguments
   args = parse_ptf_stdin()
   exec_dir = args.run_path
   data_dir = args.data_path
   template_dir = args.templates_path
   par_file = args.parameters_file
   user_pois = args.user_pois
   if (exec_dir is None) or (data_dir is None) or (template_dir is None) or (par_file is None) or (user_pois
is None):
        print("One of these parameters is missing: run_path, data_path, templates_path, par_file, user_pois")
   else:
        hours = args.hours
        group_sims = int(args.group_sims)
        kag = int(args.kagan_weight)
        tsu = int(args.mare_weight)
        seistype = args.seistype
        event_id = args.event
        args.event = data_dir + "/earlyEst/" + event_id + "_stat.json"
        files_step1, files_step2, files_step3, fig_path, intermediate_files = build_steps_dirs(exec_dir,
seistype)

        ### Building Configuration
        config_file = exec_dir + "/ptf_main.config"
        config_template = template_dir + "/Step1_config_template_mod.txt"
        build_config(config_template, config_file, data_dir, files_step2, par_file, str(kag), str(tsu),
event_id, user_pois)

        ### creation of the scenario ensemble ###
        sim_step1_events_file = step1_func(args, config_file, seistype, files_step1)
        depth_file = data_dir + "/regional_domain/bathy_grids/regional_domain_POIs_depth.dat"
        grid_file = data_dir + "/regional_domain/bathy_grids/regional_domain.grd"
        pois_file = data_dir +"/regional_domain/POIs.txt"
        sim_template_file = template_dir + "/Step2_parfile_tmp.txt"
        log_file = files_step2 + "/Step2_"+seistype+"_failed.txt"
        sim_pois_ts = exec_dir + "/Step2_ts.dat"
        conv_file = files_step3 + "conv_file.txt"
        sim_files_step2 = exec_dir +"/sim_files.txt"

        ### Preparation of the files for HySEA simulation ###
        build_structure(seistype, grid_file, hours, gpus_per_node, sim_files_step2, load_balancing_bin,
pois_file, sim_template_file, sim_step1_events_file, exec_dir)
        compss_wait_on_file(sim_files_step2)
        ptf_files = []
        sims = 0
        ### HySEA simulations ###
        with open(sim_files_step2) as f:
           for line in f:
                # load balancing
                line=exec_dir+'/'+line.strip()
                result=mpi_func(line,exec_dir)

                with open(line) as fsim:
                        for simline in fsim:
                        sims = sims + 1
                        simline = simline.strip()
                        simwdir = os.path.dirname(simline)
                        out_ts = exec_dir +"/"+simwdir+"/out_ts.nc"
                        ptf_file = exec_dir +"/"+simwdir+"/out_ts_ptf.nc"
                        ts_result = extract_ts(out_ts,depth_file,ptf_file,simwdir,result)
                        append_and_evaluate(ptf_files, ptf_file, args, config_file, files_step1,
files_step2, intermediate_files, files_step3, depth_file, log_file, sim_pois_ts, group_sims, kag, tsu,
ts_result, conv_file, fig_path)
                fsim.close()
           f.close()
        # Compute a final evaluation if final group was not multiple of group_sims
        if (group_sims == 0) or (sims % group_sims != 0):
                append_and_evaluate(ptf_files, ptf_file, args, config_file, files_step1, files_step2,
intermediate_files, files_step3, depth_file, log_file, sim_pois_ts, 1, kag, tsu, ts_result, conv_file,
fig_path)
        compress(files_step3, exec_dir+"/results_" + event_id + "_" + seistype + ".tar.gz", ptf_files)
```

12 Appendix D

UCIS4EQ original code

```
def checkPostRequest(r):
        if r.json()['response'] == 501:
        raise Exception(r.json()['result'])

class WorkflowManagerEmulator(microServiceABC.MicroServiceABC):

        # Initialization method
        def __init__(self):
        """
        Initialize the sourceType component implementation
        """

        # Obtain UCIS4EQ Services URL
        localhost = "http://127.0.0.1"
        self.url = os.getenv("UCIS4EQ_LOCATION", localhost)
        if self.url == "":
        self.url = localhost


        @staticmethod
        def compute(lalert, url):
        r = requests.post(url + ":5002/Graves-Pitarka", json=lalert)
        config.checkPostRequest(r)
        lalert["rupture"] = r.json()['result']

        # Generate the input parameter file for phase 2 in YAML format
        r = requests.post(url + ":5000/inputParametersBuilder", json=lalert)
        config.checkPostRequest(r)
        stage2InputP = r.json()['result']

        # TODO:
        # Call a service in charge of deciding the simulator code (HUB)

        # Build the Salvus input parameter file (remotely)
        sim = {}
        sim["id"] = lalert['id']
        sim["trial"] = lalert['trial']
        sim["input"] = stage2InputP
        sim["resources"] = lalert['resources']
        r = requests.post(url + ":5003/SalvusPrepare", json=sim)
        config.checkPostRequest(r)
        sim["input"] = r.json()['result']

        # Call Salvus system (or other)
        r = requests.post(url + ":5003/SalvusRun", json=sim)
        config.checkPostRequest(r)

        # Call Salvus post (or other)
        r = requests.post(url + ":5003/SalvusPost", json=sim)
        config.checkPostRequest(r)

        return lalert

        # Service's entry point definition
        @config.safeRun
        def entryPoint(self, body):
        """
        Temporal workflow manager emulator
        """

        # Just a naming convention
        event = body

        # Obtain the Event Id. (useful during all the workflow livecycle)
        r = requests.post(self.url + ":5000/eventRegistration", json=event)
```

```
        config.checkPostRequest(r)
        self.eid = r.json()['result']

        # Obtain the region where the event occured
        payload = {'id': self.eid}
        r = requests.post(self.url + ":5000/eventDomains", json=payload)
        config.checkPostRequest(r)
        domains = r.json()['result']

        # Check the region
        if not domains:
        # Set the event with SUCCESS state
        payload = {'id': self.eid, 'state': "REJECTED"}
        r = requests.post(self.url + ":5000/eventSetState",json=payload)
        config.checkPostRequest(r)
        self.eid = r.json()['result']

        raise Exception("There is not enough information for simulating the EQ in region")

        # For each found domain
        for domain in domains:
        input = {}

        # Creating the pool executor
        with concurrent.futures.ProcessPoolExecutor(max_workers=10) as executor:
                futures = []

                # Calculate the CMT input parameters
                payload = {'id': self.eid, 'region': domain['region']}
                r = requests.post(self.url + ":5000/precmt", json=payload)
                config.checkPostRequest(r)
                precmt = r.json()['result']

                # Obtain region information
                payload = {'id': domain['region']}
                r = requests.post(self.url + ":5000/eventGetRegion", json=payload)
                config.checkPostRequest(r)
                region = r.json()['result']

                event = precmt['event']
                setup = {'setup': precmt}
                payload = {'id': self.eid, 'domain': domain}
                r = requests.post(self.url + ":5000/computeResources", json=payload)
                config.checkPostRequest(r)
                compResources = r.json()['result']

                # Compute CMTs for each pair event-alert (earthquake - Agency notification)
                input = {'id': self.eid, 'base': "event_" + body['uuid']}
                input.update(setup)

                # Append the region
                input["domain"] = domain
                input["resources"] = compResources

                for a in event['alerts']:

                input['event'] = a

                r = requests.post(self.url + ":5000/cmt", json=input)
                config.checkPostRequest(r)

                cmt = r.json()['result']

                if "cmt" in a.keys():
                        cmt.update(a["cmt"])

                input['event'].update({"CMT":cmt})

                #alert = event['alerts'][a]
                #print(json.dumps(alert))

                # Determine the appropiate source for this event
                # TODO: Define the input parameters that we need for this step
                payload = {'id': self.eid }
```

```
                r = requests.post(self.url + ":5000/sourceType", json=payload)
                config.checkPostRequest(r)

                sourceType  = r.json()['result']

                # Compute source
                # TODO: Select the correct way according to the received source type

                # For each CMT in the alert
                for cmt in input['event']['CMT'].keys():

                        # For each GP defined trial
                        for slip in range(1, region['GPSetup']['trials']+1):
                        # Set the trial path
                        tags = ".".join([domain['id'], cmt, "slip"+str(slip)])

                        # Create a local alert
                        lalert = {}
                        lalert['event'] = input['event'].copy()
                        lalert['id'] = self.eid
                        lalert['trial'] = input['base'] + "/trial_" + tags
                        lalert['CMT'] = input['event']['CMT'][cmt]
                        lalert['domain'] = input['domain']
                        lalert['resources'] = input['resources']

                        if 'seed' in a.keys():
                                lalert['event']['seed'] = a['seed']

                        futures.append(executor.submit(WorkflowManagerEmulator.compute, lalert, self.url))


                for future in concurrent.futures.as_completed(futures):
                data = future.result()

        # Post-process output by generating:
        r = requests.post(self.url + ":5003/SalvusPlots", json=input)
        config.checkPostRequest(r)

        # Set the event with SUCCESS state
        payload = {'id': self.eid, 'state': "SUCCESS"}
        r = requests.post(self.url + ":5000/eventSetState", json=payload)
        config.checkPostRequest(r)
        self.eid = r.json()['result']

        # Return list of Id of the newly created item
        return jsonify(result = "Event with UUID " + str(body['uuid']) + " notified for region " +
str(domain['region']), response = 201)
```

13 Appendix E

PyCOMPSs version of the UCIS4EQ Workflow

```
class PyCommsWorkflowManager(microServiceABC.MicroServiceABC):

        # Initialization method
        def __init__(self):
        """
        Initialize -the workflow manager
        """

        # Service's entry point definition
        @config.safeRun
        def entryPoint(self, body):
        """
        PyCOMPSs workflow manager
        """

        event = body
        #print(body)

        # Set event's name
        basename = "event_" + event['uuid']

        # Obtain the Event Id. (useful during all the workflow livecycle)
        eid = register_event(event)

        # Obtain the region where the event occured
        region = compss_wait_on(get_region(eid))

        # Obtain the setup depending on the incoming event
        setup = compss_wait_on(get_setup(eid))

        if not region:
        eid = set_event_state(eid, "REJECTED")
        return "WARNING: There is not enough information for simulating the following event", 500

        # Calculate computational resources for the given domain
        resources = compute_resources(eid, region)

        # Obtain rupture generator's setup
        gpsetup = compss_wait_on(graves_pitarka_setup(eid, region, setup))

        # Calculate the CMT input parameters
        precmt = build_cmt_input(eid, region, resources, setup)
        # Compute alerts
        all_results = []
        for alert in event['alerts']:
        # Calculating CMTs
        cmts = compss_wait_on(calculate_cmt(alert, eid, region, precmt))
        # For each calculated or provided CMT
        for cmt in cmts.keys():
                # For each GP defined trial
                for slip in range(1, gpsetup['trials']+1):
                # Set the trial path
                path = "event_" + event['uuid'] + "/trial_" + ".".join([cmt, "slip"+str(slip)])
                all_results.append(launch_simulation(eid, alert, path, cmts[cmt], region, gpsetup, resources,
ensemble))
                # break
                # break
        #break
        result = compss_wait_on(launch_post_swarm(eid, region, basename, resources, all_results))
        eid = set_event_state(eid, "SUCCESS")

        # Return list of Id of the newly created item
        return jsonify(result = "Event with UUID " + str(body['uuid']), response = 201)

def launch_simulation(eid, alert, path, data, region, gpsetup, resources, ensemble):
        # Call Graves-Pitarka's rupture generator
        rupture = compute_graves_pitarka(eid, alert, path, data, region, gpsetup, resources, ensemble)
        # Call input parameters builder
        inputs = build_input_parameters(eid, alert, data, rupture, region, resources, gpsetup, ensemble)
```

```
        # Build the Salvus input parameter file (remotely)
        salvus_inputs = build_salvus_parameters( eid, path, inputs, resources)
        # Build the Salvus input parameter file (remotely)
        salvus_result = run_salvus( eid, path, salvus_inputs,resources)
        # Call Salvus post
        return run_salvus_post(eid, salvus_result, path, resources)

def launch_post_swarm(eid, region, basename, resources, all_results):
        # Call postprocessing swarm
        output_swarm = run_salvus_post_swarm(eid, basename, resources)

        # General post-processing for generating plots
        return run_salvus_plots(eid, output_swarm, region, basename, resources)

@http(request="POST",   resource="eventRegistration",   service_name="microServices",   payload="{{event}}",
produces='{"result" : "{{return_0}}" }')
@task(returns=1)
def register_event(event):
        pass


@http(request="POST", resource="eventRegion", service_name="microServices", payload='{ "id" : {{event_id}}
}', produces='{"result" : "{{return_0}}" }')
@task(returns=1)
def get_region(event_id):
        pass

@http(request="POST", resource="eventSetup", service_name="microServices", payload='{ "id" : {{event_id}},
"ensemble" : "{{ensemble}}" }', produces='{"result" : "{{return_0}}" }')
@task(returns=1)
def get_setup(event_id, ensemble):
        pass

@http(request="POST", resource="eventSetState", service_name="microServices", payload='{ "id" : {{event_id}},
"state": "{{state}}" }', produces='{"result" : "{{return_0}}" }')
@task(returns=1)
def set_event_state(event_id, state):
        pass

@http(request="POST", resource="cmtSeisEnsMan", service_name="microServices", payload='{ "id" : {{event_id}},
"region": {{region}},  "resources": {{resources}}, "setup": {{setup}} }', produces='{"result" : "{{return_0}}"
}')
@task(returns=1)
def input_seisensman(event_id, region, resources, setup):
        pass

@http(request="POST",  resource="precmt",  service_name="microServices",   payload='{  "id"  :  {{event_id}},
"region": {{region}}, "resources": {{resources}}, "setup": {{setup}} }', produces='{"result" : "{{return_0}}"
}')
@task(returns=1)
def build_cmt_input(event_id, region, resources, setup):
        pass

@http(request="POST", resource="cmt", service_name="microServices",  payload='{ "event" : {{alert}}, "id" :
{{event_id}}, "region" : {{region}}, "setup" : {{precmt}} }', produces='{"result" : "{{return_0}}"}')
@task(returns=1)
def calculate_cmt(alert, event_id, region, precmt):
        pass

@http(request="POST",   resource="computeResources",   service_name="microServices",   payload='{   "id"   :
{{event_id}}, "region": {{region}} }', produces='{"result" : "{{return_0}}" }')
@task(returns=1)
def compute_resources(event_id, region):
        pass

@http(request="POST",  resource="preGraves-Pitarka",  service_name=preGP_service_name,  payload='{  "id"  :
{{event_id}}, "region": {{region}}, "setup": {{setup}} }', produces='{"result" : "{{return_0}}" }')
@task(returns=1)
def graves_pitarka_setup(event_id, region, setup):
        pass

@http(request="POST", resource="Graves-Pitarka", service_name="slipgen", payload='{ "event" : {{alert}}, "id"
: {{event_id}}, "CMT" : {{cmt}}, "trial" : "{{path}}", "region": {{region}}, "setup" : {{setup}}, "resources"
: {{resources}} , "ensemble" : "{{ensemble}}" }', produces='{"result" : "{{return_0}}"}')
```

```
@task(returns=1)
def compute_graves_pitarka(event_id, alert, path, cmt, region, setup, resources, ensemble):
        pass

@http(request="POST", resource="inputParametersBuilder", service_name="microServices", payload='{ "id" :
{{event_id}}, "event" : {{alert}}, "CMT" : {{cmt}}, "rupture" : {{rupture}}, "region" : {{region}}, "resources"
: {{resources}}, "setup" : {{setup}}, "ensemble" : "{{ensemble}}" }', produces='{"result" : "{{return_0}}"}')
@task(returns=1)
def build_input_parameters(event_id, alert, cmt, rupture, region, resources, setup, ensemble):
        pass

@http(request="POST", resource="SalvusPrepare", service_name="salvus", payload='{ "id" : {{event_id}},
"trial" : "{{trial}}", "input" : {{input}}, "resources" : {{resources}} }', produces='{"result" :
"{{return_0}}"}')
@task(returns=1)
def build_salvus_parameters(event_id, trial, input, resources):
        pass


@http(request="POST", resource="SalvusRun", service_name="salvus", payload='{ "id" : {{event_id}}, "trial" :
"{{trial}}", "input" : {{input}}, "resources" : {{resources}} }', produces='{"result" : "{{return_0}}"}')
@task(returns=1)
def run_salvus(event_id, trial, input, resources):
        pass

@http(request="POST", resource="SalvusPost", service_name="salvus", payload='{ "id" : {{event_id}}, "trial"
: "{{trial}}", "resources" : {{resources}} }', produces='{"result" : "{{return_0}}"}')
@task(returns=1)
def run_salvus_post(event_id, salvus_result, trial, resources):
        pass

@http(request="POST", resource="SalvusPostSwarm", service_name="salvus", payload='{ "id" : {{event_id}},
"base" : "{{base}}", "resources" : {{resources}} }', produces='{"result" : "{{return_0}}"}')
@task(returns=1)
def run_salvus_post_swarm(event_id, base, resources):
        pass

@http(request="POST", resource="SalvusPlots", service_name="salvus", payload='{ "id" : {{event_id}}, "region"
: {{region}}, "base" : "{{base}}", "resources" : {{resources}} }', produces='{"result" : "{{return_0}}"}')
@task(returns=1, results=COLLECTION_IN)
def run_salvus_plots(event_id, salvus_post_results, region, base, resources):
        pass
```

14 Appendix F

PyCOMPSs workflows for MLESmap

- Model Selection:

```
def write_results(results_path, cv_results, best_model):
        pd_df = pd.DataFrame.from_dict(cv_results)
        pd_df.to_csv(results_path +'/pd.csv')
        best_model.save_model(results_path + '/model.dat', save_format='pickle')

def model_selection(input_dataset, results_path, parameters):
        df = load_txt_file(input_dataset ,discard_first_row=True, col_of_index=True,block_size=(133334, 9))
        Data_X_arr = df[:, 0:8]
        Data_Y_arr = df[:, 8:9]
        scaler_X = MinMaxScaler(feature_range=(0, 1))
        scaler_X.fit(Data_X_arr)
        scaler_y = MinMaxScaler(feature_range=(0, 1))
        scaler_y.fit(Data_Y_arr)
        x_ds_train, x_ds_test, y_ds_train, y_ds_test = train_test_split(Data_X_arr, Data_Y_arr)
        x_ds_test = x_ds_test.rechunk((100000, 8))
        y_ds_test = y_ds_test.rechunk((100000, 1))
        x_train = scaler_X.transform(x_ds_train)
        x_test = scaler_X.transform(x_ds_test)
        y_train = scaler_y.transform(y_ds_train)
        y_test = scaler_y.transform(y_ds_test)
        rf = RandomForestRegressor(n_estimators=15, try_features = 'third')
        searcher = GridSearchCV(rf, parameters, cv=3)
        searcher.fit(x_train, y_train)
        write_results(results_path, searcher.cv_results_, searcher.best_estimator_)

if __name__ == "__main__":
   dataset_file = str(sys.argv[1])
   results_path = str(sys.argv[2])
   f = open(sys.argv[3])
   parameters = json.load(f)
   model_selection(dataset_file, results_path, parameters)
```

- Inference:

```
def load_scalers(x_scaler_file, y_scaler_file):
        scaler_X = MinMaxScaler(feature_range=(0, 1))
        scaler_X.load_model(x_scaler_file)
        scaler_y = MinMaxScaler(feature_range=(0, 1))
        scaler_y.load_model(y_scaler_file)
        return scaler_X, scaler_y

def load_rf_model(model_file):
        rf = RandomForestRegressor(max_depth=15,n_estimators=15,try_features='third',random_state=0)
        rf.load_model(model, load_format="pickle")
        return rf

def predict(df, rf, scaler_X, scaler_y, result_file):
        x_test = scaler_X.transform(df[:, 0:8])
        y_pred = scaler_y.inverse_transform(rf.predict(x_test))
        y_pred = y_pred.collect()
        np.savetxt(result_file,y_pred)

if __name__ == '__main__':
        scaler_X,scaler_Y=load_scalers(sys.argv[1], sys.argv[2])
        rf = load_rf_model(sys.argv[3])
        df = load_txt_file(sys.argv[4], discard_first_row=True, col_of_index=True,block_size=(300, 9))
        predict(df, rf, scaler_X, scaler_Y, sys.argv[5])
```

15 Appendix G

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
MÉXICO SSN
SERVICIO SISMOLÓGICO NACIONAL
GEOFISICA UNAM
INSTITUTO DE GEOFÍSICA, UNAM
CIRCUITO DE LA INVESTIGACIÓN CIENTÍFICA S/N, CIUDAD UNIVERSITARIA, DELEGACIÓN COYOACÁN, C.P. 04510, MÉXICO D.F.
Tel.: 5622 2222 Ext. 38700 www.ssn.unam.mx contacto@sismologico.unam.mx

Ciudad Universitaria a 29 de septiembre de 2023

Dr. Josep de la Puente
Wave Phenomena Group
Barcelona Supercomputing Center
Jordi Girona, 29, 08034, Barcelona

Mexico, 23 October 2023

To whom it may concern

This letter confirms my interest in using the Urgent Computing Integrated Services for Earthquake workflow developed in the eFlows4HPC project. As the Operational Mexican Seismological Service we are monitoring the earthquakes that affect the Mexican country 24/7 and we are interested in validating the outcomes of UCIS4EQ for large magnitude earthquakes and testing the workflow under situations analogous to those of an operational service. Moreover, as a part of the User and Industrial board of the Center of Excellence for Exascale in Solid Earth (ChEESE-2P) we wish to contribute towards the successful development of novel applications that may be of interest to our institution.

Yours sincerely,

Arturo Iglesias

SSN Scientific Responsible