Automatic, Efficient and Scalable Provenance Registration for FAIR HPC Workflows

Raül Sirvent
Barcelona Supercomputing Center
raul.sirvent@bsc.es

Javier Conejero
Barcelona Supercomputing Center
javier.conejero@bsc.es

Francesc Lordan
Barcelona Supercomputing Center
francesc.lordan@bsc.es

Jorge Ejarque
Barcelona Supercomputing Center
jorge.ejarque@bsc.es

Laura Rodríguez-Navas
Barcelona Supercomputing Center
laura.rodriguez@bsc.es

José M. Fernández
Barcelona Supercomputing Center
jose.m.fernandez@bsc.es

Salvador Capella-Gutiérrez
Barcelona Supercomputing Center
salvador.capella@bsc.es

Rosa M. Badia
Barcelona Supercomputing Center
rosa.m.badia@bsc.es

***Abstract*—Provenance registration is becoming more and more important, as we increase the size and number of experiments performed using computers. In particular, when provenance is recorded in HPC environments, it must be efficient and scalable. In this paper, we propose a provenance registration method for scientific workflows, efficient enough to run in supercomputers (thus, it could run in other environments with more relaxed restrictions, such as distributed ones). It also must be scalable in order to deal with large workflows, that are more typically used in HPC. We also target transparency for the user, shielding them from having to specify how provenance must be recorded. We implement our design using the COMPSs programming model as a Workflow Management System (WfMS) and use RO-Crate as a well-established specification to record and publish provenance. Experiments are provided, demonstrating the run time efficiency and scalability of our solution.**

***Index Terms*—Provenance, Reproducibility, Replicability, Scientific Workflows, FAIR, High Performance Computing, COMPSs, RO-Crate**

I. Motivation

The volume of data gathered to be analysed, computed or treated in many different domains keeps continuously growing. Even more, if we consider the increase in computing capacities (i.e., the more computing we have available, the more data we want to gather to be analysed). Besides, when many results are generated, the need for adding extra metadata on how an experiment or an analysis was conducted becomes much more important, if we want to understand how they were achieved (i.e., the more results we have, the more important metadata becomes). This can be seen specially in domains such as Big Data Analytics, IoT and Scientific Workflows. It is also a fact that results of such analyses have been traditionally shared in research by just providing a numerical and textual description of the experiments in a scientific paper, where readers could not easily verify those results, leading to what we know these days as the reproducibility crisis [1].

With the objective of dealing with these problems, the *Provenance* field has been targeting to register and associate metadata to existing or newly generated data in order to improve reproducibility or replicability of scientific results. Reproducibility to allow third parties to re-execute an experiment exactly the way it was originally conducted (to verify results). Replicability to be able to apply the same procedures specified in the experiment to a new input set, obtaining different results. The literature also presents the concept of FAIR data (Findable Accessible Interoperable and Reusable) [2] to solve these previously mentioned problems.

We find many different existing systems to register provenance, some of which are thought to be directly consumed by humans (i.e., with visual tools, like the EBRAINS Data and Knowledge Service[1]). Others are oriented to be consumed by machines, with a specific focus on interoperability (RDF[2], OWL[3], PROV [3]). Hand-made provenance registration using visual tools has the clear drawback of scalability, since a drawing can become very complex when the number of involved entities is very large. On the other hand, using directly RDF and OWL may be difficult for non-computer scientists.

In this paper, we propose to create a lightweight interoperable provenance registration method for scientific workflows in the High Performance Computing (HPC) domain. In HPC environments, performance is a mandatory dimension, since

This work has been supported by the Spanish Government (PID2019-107255GB-C21), by Generalitat de Catalunya (contract 2017-SGR-01414) and the EU's Horizon research and innovation programme under Grant agreement No 101058129 (DT-GEO). Also, it has been contributed in the CECH project, co-funded with 50% by the European Regional Development Fund under the framework of the ERFD Operative Programme for Catalunya 2014-2020, with a grant of 1.527.637,88€. LRN, JMF and SCG are partly supported by INB Grant (PT17/0009/0001 - ISCIII-SGEFI / ERDF), and their work received funding from the EU's Horizon 2020 research and innovation programme under grant agreements EOSC-Life No 824087, and EJP RD No 825575. All URLs in footnotes have been accessed on October 2022.

[1]https://ebrains.eu/services/data-and-knowledge/
[2]https://www.w3.org/RDF/
[3]https://www.w3.org/2001/sw/wiki/OWL

 DOI 10.1109/WORKS56498.2022.00006

the cost of supercomputers is high, and their resources must always be used in the most effective way. Therefore, our provenance registration must be efficient and scalable. Efficient to avoid including run time overheads when recording provenance. Scalable to be able to deal with large workflows (more common in supercomputers). We also target our registration to be automatic, meaning that no specific annotations must be provided by the user to indicate what needs to be recorded, or at which level of granularity. We will use the COMPSs programming model to implement such registration mechanism, and demonstrate its properties with two real use case scenarios that belong to two different domains, showing that our solution is domain-agnostic. Our workflow runs are made FAIR by being published in the WorkflowHub registry (Findable and Accessible), while leveraging the use of RO-Crate, that makes them Interoperable and Reusable.

The rest of this paper is structured as follows: Section II reviews existing literature on provenance, Section III presents the needed background that will be used to design our solution (described in detail in Section IV). At the end, Section V presents results for two use cases from life and earth sciences domains. And finally, Section VI reviews the main contributions of this paper and proposes future work.

II. Related Work

Ideas about provenance recording for information systems have been around for many years, as shown by Buneman et al. [4] in the domain of databases during the early 2000s. Anyway, although provenance concepts are not new, they are currently being applied to many new areas.

The provenance literature may be a bit misleading term-wise. A very good attempt of establishing a common terminology throughout provenance literature is provided by Herschel et al., [5], where workflow provenance is particularly explored. However, during recent years, the terms have continued evolving while being re-adapted to different fields, using new and related wording such as Data Lineage, Data Curation, Data Governance. We consider all these new terms included in a semantically broader concept of *Provenance*. In addition, the most recent well-established FAIR [2] also targets reproducibility, as registering provenance does. Considering the classification made at [5], the provenance registration targeted in our paper includes information system provenance (in our case, the general parameters of a process), workflow provenance (the main workflow source file) and data provenance (the source code of a task in the workflow that generates a specific data asset). However, our provenance is not related to databases, as it is assumed in many papers when the term *data provenance* is used.

As discussed in [6], in the late 2000s provenance for workflows were early explored, and both *prospective provenance* (i.e., the workflow definition) and *retrospective provenance* [7] (i.e., steps and environment needed to generate a data asset) were targeted. While the first one was covered by their own workflow language definition on each case, the second was commonly solved by gathering general information of a specific run of a workflow, such as the user who ran it, execution time, and some other basic information or statistics. Then, many of these systems stored their retrospective provenance in a separate database, with the goal of later querying it. Although most of these frameworks used their own provenance format representation, some of these approaches even considered using Semantic Web technologies such as RDF or OWL to achieve certain interoperability, which may be complex to be used directly for non-computer science specialists.

During the early 2010s, and as the need was seen in previous papers, standardisation and interoperability activities emerged, like in the W3C Provenance Working Group with the resulting PROV family of documents [3]. In particular, the FAIRness idea gained momentum in the upcoming years since the publication of [2], and we see these days many research projects and papers concerned about FAIRness of their data. Even some conferences have started to request data assets that can prove the results obtained in a publication, in order to ensure the paper results can be *reproducible* by other researchers. The ideas established in such initiatives have recently derived in projects such as the Research Object Crate [8], that targets to establish a lightweight approach to package research data together with their metadata. RO-Crate will be described in more detail in Section III-B.

With respect to current workflow environments that are able to generate provenance of some kind, Nextflow [9] includes a *log* command[4], which enables to return some information about the workflow execution that can later be used to manually build a provenance report. Similarly, Snakemake [10] has the *report* and *summary* functionalities, where different bits of provenance information are returned. In both cases, some manual annotations or template definitions from the user are expected, and a own format is used (jeopardizing interoperability). The Common Workflow Language project (CWL) produced the CWLProv [11] format, and implemented a hierarchical provenance registration framework using Research Objects (ROs), BagIt and PROV. In their paper, runs with provenance activated cause quite an overhead in the experiments. Besides, no scalability studies with large workflows are included (i.e., thousands of tasks).

Our lightweight provenance registration approach is transparent to users and scalable, as required in [12] to achieve FAIR computational workflows. To the best of our knowledge, none of the existing environments have been able to demonstrate automation (no provenance-related annotations), efficiency (no run time overhead) and scalability (large workflows and number of data assets) all at the same time for provenance registration, as we do in this paper.

III. Background

The solution we present at Section IV is mainly based on two existing frameworks: the COMPSs programming model and the RO-Crate specification. This Section provides a brief introduction to both.

[4]https://training.seqera.io/#_execution_provenance

A. COMPSs

COMP Superscalar (COMPSs) [13] is a task-based programming model that focuses on making easier the development of parallel applications for distributed computing. Its syntax is based on the use of annotations to identify those methods that will become tasks at execution time and in a small API for synchronization.

COMPSs supports Java as native language, and Python (PyCOMPSs [14]) and C through language bindings. Listing 1 shows a sample Python task annotated with the `@task` decorator. Tasks are the parallelism unit. Every time an annotated method (task) is invoked from the main program, the COMPSs runtime creates a node in a task-graph and looks for data dependencies with previous existing tasks. The task annotation includes directionality clauses (i.e., `IN` in the example) that enable to infer the data dependencies at runtime. These directionality clauses allow the runtime to control how data changes during the workflow execution (i.e., `IN` indicates data used by the task, `OUT` indicates data created in the task, `INOUT` indicates data modified in the task). Each node in the graph denotes a task, and edges between them denote data dependencies between the connected tasks. The runtime is then able to exploit the potential parallelism of the task-graph by scheduling those tasks that do not have data dependencies between them, and it is also able to execute in an asynchronous fashion, being able to start new tasks once their predecessors end.

```
@task(data:IN, returns=dict)
def wordCount(data):
    partialResult = {}
    for entry in data:
        if entry not in partialResult:
            partialResult[entry] = 1
        else:
            partialResult[entry] += 1
    return partialResult
```

Listing 1: Sample task code

The syntax is completed with a small set of API calls, mainly focusing on synchronization. The COMPSs runtime handles all data transfers automatically, by moving data on-demand between the computing nodes of the cluster or distributed computing infrastructure. However, when a task result is needed in the main program, we need to wait for the task to end and data needs to be sent (synchronized) to the computing node where the main program is executed. Listing 2 shows a sample PyCOMPSs code with a synchronization call. The methods `wordCount` and `merge_two_dicts` are tasks and their invocations are executed in parallel and in a distributed fashion. In order to get the final `result`, the synchronisation call waits for all tasks to finish and collects the last value of `result` to make it available to the main program.

The described syntax is powered by the COMPSs runtime, which is organized as a set of components providing the different required functionalities. It is deployed following the master-worker paradigm, with one computing node acting as master and multiple computing nodes acting as workers. The master runs the user code (with the decorators and invocations to the API) together with the runtime, which is the entity in charge of adding nodes (tasks) and edges (data dependencies) in the task-graph, as previously explained.

```
result = {}
for block in read_word_by_word(pathF, sizeB):
    presult = wordCount(block)
    merge_two_dicts(result, presult)
result = compss_wait_on(result)
```

Listing 2: Sample PyCOMPSs code with synchronisation API

As soon as part of the tasks have been generated, the COMPSs scheduler starts to select ready to run tasks and assigns them to workers. The runtime also takes care of doing the required data transfers (i.e., files, objects, scalars, etc.) between the different computing nodes, in such a way that the data required by each task is there at execution time. To reduce the number of data transfers, the scheduler takes data locality into account.

The task parameters can be both objects in memory or files, and a special data type are data collections, which enable to detect data dependencies between tasks that operate on individual objects of a collection and others that operate with the whole object. Recently, the stream data type was also added, which enables to define dependencies between tasks that operate with streams of data [15].

The COMPSs runtime can be deployed in different types of infrastructures: clouds, large clusters (or supercomputers) and container managed clusters [16]. In the case of clusters or supercomputers managed with job schedulers, the whole COMPSs application is queued as a single allocation of a set of computing nodes. Once the allocation is obtained, the COMPSs runtime selects one of the computing nodes to act as master and the rest will become workers. Different types of container environments are supported, while the more commons are Docker or Singularity in the case of supercomputers.

With regard fault tolerance, a mechanism at task level is provided, where the programmer can indicate in a decorator the behavior to implement in case of task failure (i.e., ignore the failure of the task and continue, stop the whole workflow, etc.) [17]. In addition, a checkpointing mechanism at task level has been implemented, which enables to recover a failed execution from the last checkpointed task.

COMPSs source files are available on GitHub[5], and the provenance capabilities described in this paper have been included in version 3.0.

B. RO-Crate

Many computational analyses nowadays suffer from a lack of reproducibility and replicability because critical components are not usually published, archived or well-tracked. In this scenario, the *Research Object (RO)*[6] was proposed to improve the reproducibility and replicability of experimental

[5]https://github.com/bsc-wdc/compss
[6]https://www.researchobject.org/

results, since ROs allow to semantically relate all the digital objects which were involved to obtain research results, including identifiers, provenance, relations or annotations. The *Research Object Crate (RO-Crate)* [18] specification describes the Research Objects in a structured language and is also an evolution from DataCrate[7], which details a simple method to describe and aggregate research data with associated metadata.

Compared to the current standard academic publishing process, usually a PDF with a couple of supplemental materials or the STAR Methods[8], an RO-Crate is a machine-readable bundle capable of communicating an RO's diverse digital and physical resources. It can include objects such as data (raw and processed), computational workflows and scripts, results (graphs, derived data), metadata, etc. In our work, RO-Crates include COMPSs applications (see Section III-A).

The context of our data provenance (see Section IV), the entities involved in its generation, and its reproducibility and replicability are all described as associated metadata in an accessible and practical manner. RO-Crate uses a JSON-LD representation (a structured form of JSON that can represent linked and semantic data graphs) to declare the structured metadata by using vocabularies, mainly published in Schema.org[9]. Thus, we use the valid and graph-structured *RO-Crate JSON-LD* that includes:

- *RO-Crate Metadata File:* A file (*ro-crate-metadata.json*) describing the RO-Crate, its content, and related metadata using linked data, as well as providing authors, organizations and licenses.
- *Root Data Entity:* A directory identified by the presence of the RO-Crate Metadata File at the root (the RO-Crate Root) that provides metadata about other files and directories represented by Data Entities. It represents the RO as a dataset and its minimal requirements are *name*, *description* and *datePublished*, as well as a contextual entity identifying its *license*.
- *Data Entities:* Files or directories located within the RO-Crate Root using the *hasPart* property, and web resources or restricted data recognised by absolute IRIs, including Persistent Identifiers (PIDs). Data entities are further described with contextual entities.
- *Contextual Entities:* Non-digital elements (e.g., a person or an organization) or conceptual descriptions as metadata, like contact information.

RO-Crates can be used as general-purpose containers for arbitrary data and open metadata exchange. Such freedom, in practice, can hinder the interoperability of systems able to produce or consume RO-Crates. To correct this, *RO-Crate profiles* are promoted as an extensible mechanism that allows defining and describing the minimal set of conventions, types, properties, and restrictions to be fulfilled to allow interoperability within a given domain, application, or framework. One of these profiles is the *Workflow RO-Crate profile*, which is a specialization of RO-Crate on workflows for capturing the provenance of computational workflow executions. This profile strictly conforms to the more generic Bioschemas *ComputationalWorkflow*[10] profile as data entity.

The Workflow RO-Crate profile is used in this paper for packaging, typing, and annotating a COMPSs workflow's constituent files. The generated RO-Crate describes the authors, license, diagram previews, and a list of the workflow's inputs and outputs. We utilised the RO-Crate Python library [19] (ro-crate-py) to build this RO-Crate, and the WorkflowHub registry [20] to publish the workflows. WorkflowHub is a registry for describing, sharing and publishing scientific computational workflows, and it is compliant with the RO-Crate specification. By simply providing the generated crate (i.e., the package of files with a corresponding workflow run), WorkflowHub is able to automatically import and publish a workflow, facilitating also the generation of a corresponding DOI.

The RO-Crate specification has recently become the confluence of a growing community of developers and users from very different disciplines. It is being applied in research domains such as Bioinformatics, Regulatory Science and Cultural Heritage. Furthermore, many tools from its ecosystem, such as Describo[11], and libraries such as the previously mentioned ro-crate-py, are easing the adoption of the specification and promoting relevant developments, including WorkflowHub, UTS Cultural Datasets[12] or the WfExS-backend[13].

IV. Design and implementation of efficient provenance gathering

As it has been briefly seen in Section I, we established a set of requirements for enabling FAIR HPC workflows that are going to influence the design of our solution: *automatic, efficient and scalable*. More in detail:

- Our data provenance registration must provide support to HPC workflows, but not be limited to them.
- It must enable reproducibility and replicability, specifically for workflows.
- Automatic provenance registration is wanted, to shield users about having to register their data assets and workflow processes by hand or with annotations.
- Registration must be scalable, to support very big workflows (e.g., thousands of inputs, outputs and tasks).
- The provenance representation format must be simple to use, but powerful enough to represent complex workflows (i.e., large number of tasks and input/output data).
- Provenance registration must be efficient, and not add any significant overheads that can increase the workflow makespan.

After a thorough review of the literature and current experiences in provenance (see Section II), we selected ***RO-Crate*** as the standard to be used for registering the workflow execution.

[7]https://github.com/UTS-eResearch/datacrate.git
[8]https://www.cell.com/star-methods
[9]https://schema.org/
[10]https://bioschemas.org/profiles/ComputationalWorkflow/1.0-RELEASE
[11]https://github.com/Arkisto-Platform/describo
[12]https://arkisto-platform.github.io/case-studies/uts-cultural/
[13]https://github.com/inab/WfExS-backend

RO-Crate has been described in detail in Section III-B and, as it has been seen, many of its features match our requirements. It is a simple format (JSON-LD, simpler than any existing formats to register provenance), but very powerful, as the different profiles that can be created on top of it demonstrate. In our case, we will be compliant with the workflow profile[14], enabling reproducibility and replicability of workflows. Automatic generation of RO-Crates can be achieved by using some of the tools that have been built as an ecosystem around the standard (as detailed here[15]), and scalability is also covered, since with RO-Crate there are no limits in terms of how many software bundles or input/output files can be represented (as it could happen in a graphical registration tool). The workflow runs will be published in the RO-Crate compliant WorkflowHub registry[16], so they can be searched and queried for.

A. Registered data assets using COMPSs

In order to implement our idea of automatically, efficiently and scalably generating provenance for HPC workflows, we will use the COMPSs programming model (see Section III-A). This is not only because it is our previous work, but also due to its strong capabilities regarding workflows on HPC environments, and its efficiency in handling very big workflows that can have large number of task nodes and input/output parameters. Besides, both COMPSs and RO-Crate are domain-agnostic, leading to a solution that can be applied to any field.

The registration of provenance information of COMPSs workflows needs to target not only the data assets used/generated by COMPSs (i.e., the input and output files or parameters) but also the source code of the application that generates the workflow. Currently, COMPSs supports applications written in Java, C/C++ and Python, and we have selected the Python binding (PyCOMPSs [14]) to implement provenance capture. However, the same capabilities could be achieved for Java and C/C++ applications, considering the specific particularities of each language, which we leave as future work.

The main decision to be taken when using RO-Crate to register a COMPSs workflow execution is the data assets needed to be directly included in the *crate* (i.e., the package that includes all files resulting from recording provenance). A COMPSs application is composed by its source code files (this is, the programmed application) and the input files and parameters passed to the application, so it can be run. Thus, if we want to re-execute a COMPSs program, all this information needs to be included in the RO-Crate. However, many scientific workflows use very big files as inputs, and to avoid packaging them in the RO-Crate and having big data movements between environments, we will add them as URIs, so users know where to find them to reproduce an execution. This approach is commonly used in the life sciences domain, where online catalogs of commonly used files (e.g., the EGA Archive[17]) are provided, and applications that process them only need to refer to the URI where they can be found for downloading.

Our URI minting is influenced by the HPC case (using a supercomputer), and the fact that COMPSs does not accept URIs as file parameters (only local files). The URI points to a hostname, and the path where a file can be found in its local file system. A new user willing to utilize the corresponding URIs will need to have prior access granted to both hostname and paths specified. URIs in this scenario are not meant to be directly accessed, but placeholders to indicate users that the file can be found in a particular host, in a specific path.

Including the source files of the COMPSs application in the provenance recording guarantees *replicability*, since it will allow to execute the same application using different inputs, and generating different outputs. When we also add inputs and outputs together with the source code, we are ensuring that the results of a workflow execution can be verified by third parties (*reproducibility*), using the same supercomputer.

B. COMPSs runtime modifications

The COMPSs runtime is able to generate a workflow from a sequential piece of code, and analyse data dependencies between the tasks generated (i.e., the function calls in the code), therefore, it contains all the information we need to automatically generate an RO-Crate document. Simple parameters used in the application (e.g., scalars, strings, etc.) can be either passed in the CLI, or coded in the main application. As will be seen, including a `command_line_parameters.txt` file we can record CLI introduced parameters. In addition, the source files of the application will be included in the crate, thus, covering both cases.

The biggest assets exchanged between computing nodes are files, so, one of our main targets will be to correctly register the files that the workflow has used as inputs, and generated as outputs. We have modified the COMPSs runtime to automatically log any access to a file in a workflow's task, when a specific *–provenance* flag is activated. We register the URI of the file used, together with the direction of the parameter: if the file is an `IN` (only read in the task), `OUT` (newly generated in the task), or `INOUT` (read, but also modified in the task). This lightweight information registration approach at run time is key to achieve efficiency. Listing 3 shows an example of a `dataprovenance.log` file, where accesses to files are stored.

```
3.0.rc2206
lysozyme_in_water.py
App_Profile.json
file://s03r1b01-ib0/home/bsc19/dataset/2hs9.pdb IN
file://s03r1b01-ib0/home/bsc19/output/2hs9.gro OUT
file://s03r1b01-ib0/home/bsc19/output/2hs9.top OUT
...
```

Listing 3: Registered file accesses at dataprovenance.log

We can notice in the log file that the first three lines provide information that are not files:

[14]https://www.researchobject.org/ro-crate/profiles.html
[15]https://www.researchobject.org/ro-crate/tools/
[16]https://workflowhub.eu/
[17]https://ega-archive.org/

- The specific COMPSs version used (3.0.rc2206): this information is extremely relevant, because different versions of the runtime could implement some COMPSs features differently, which could affect the replicability of the workflow.
- The name of the file with the *main()* method: since applications can be programmed using many files, we need to know which one contains the main program.
- Name of the file containing the application profile (`App_Profile.json`): we allow the user to set the application profile file name, thus we register it here. The details on profiling will be discussed later.

There are other pieces of information that we deem as important to record provenance in COMPSs, and can be also automatically gathered:

- Command line parameters: as with any application, in a COMPSs application some parameters can be passed through the command line interface (CLI) when running a workflow. They need to be recorded to ensure reproducibility. We have created a text file that stores them, which is included in the crate (`command_line_parameters.txt`).
- Graphical representation of the workflow: the image of the workflow is a valuable piece of information for users, since it can help to understand the application and its parallelism chances. We enable the option in the COMPSs runtime to generate a DOT[18] file (graph description language from GraphViz) that can be later converted to a PDF file with the workflow image, and we add it to the crate (`generated_graph.pdf`).
- Application profiling: COMPSs can return information on which type and how many tasks have been executed on each resource, and statistics about their run time. We include this information in a JSON file named by default `App_Profile.json`.

C. Information provided by users

While with all the elements previously mentioned we have all the information needed to record provenance application-wise, there are certain pieces of information that cannot be automatically obtained from the execution of a COMPSs workflow. Information such as long name of the application, detailed description, license, source files forming the application, authors, and their related institutions is hard to be transparently obtained, while it is mandatory to be defined in an RO-Crate. To solve this, we define an extra YAML file `ro-crate-info.yaml` where users can specify all these terms that will be added to the RO-Crate. An example of this YAML is shown in Listing 4. Notice the use of YAML lists for authors, and the list used for the source files of the application, which provides freedom both for the number of authors and source files that can be specified.

[18]https://graphviz.org/doc/info/lang.html

```
COMPSs Workflow Information:
 name: COMPSs Matrix Multiplication
 description: Hypermatrix size 2x2 blocks
 license: Apache-2.0
 files: [matmul_files.py, matmul_tasks.py]
Authors:
 - name: Raül Sirvent
   e-mail: Raul.Sirvent@bsc.es
   orcid: https://orcid.org/0000-0003-0606-2512
   organisation_name: Barcelona Supercomputing
   ↪ Center
   ror: https://ror.org/05sd8tv96
```

Listing 4: YAML file ro-crate-info.yaml example

D. RO-Crate generation post-process

Once the COMPSs application has ended its execution and the file accesses have been registered, we have all the information we need to generate the provenance, so the next step is to process it all to obtain the RO-Crate that represents this workflow run. We have created a post-processing script in Python that iteratively processes all the information available in the `ro-crate-info.yaml` and the `dataprovenance.log`, and takes advantage of the ro-crate-py [19] library (we have used version 0.6.1) to create the associated RO-Crate. The fact that this metadata generation script is done in a separated post-process is a clear advantage when execution time is a concern: the workflow can run efficiently while only registering file accesses, and the RO-Crate creation script can be invoked by the user later, at any convenient time. More about the efficiency of this process will be discussed in Section V.

One of the special characteristics of this script is that it is able to automatically (without user intervention) identify what are *true inputs* of the workflow: only IN files that have not been generated as OUT in a previous task are considered inputs of the overall workflow. If a task generates an OUT file that is later used as an IN, we are in the case of an intermediate file. All OUT files need to be considered final results of a workflow, no matter if they are intermediate files, because they will remain unless the user deletes them explicitly. The rest of File type directions provided by COMPSs (INOUT, CONCURRENT, COMMUTATIVE) are also supported, and even File collections can be used (see the COMPSs user manual[19] for more details).

Our RO-Crate generation post-processing script adds to the RO-Crate both files that will be directly copied in the crate (as mentioned earlier), but also files referenced as URIs, which point to the machine name and path where a file can be found (see previous Listing 3). For online catalogs or open access repositories, the URL may be directly accessed but, in our particular case, we wanted to support the supercomputing case with shared file systems. The URI of a supercomputer will be only accessible for the users that can log into that machine, and have permission rights to access the path specified.

It is also important to highlight that, our post-processing script registers extra context information for each file ref-

[19]https://compss-doc.readthedocs.io/

erenced in the RO-Crate, whether it is physically included in the crate or not. The terms `contentSize` and the `sdDatePublished` are specially important to ensure files have not been altered, since they register the file size and its last modification time, respectively. Although it is currently not mandatory in the RO-Crate specification, file checksums could be added to ensure the exact same files are used when reproducing the workflow. For now, we did not add them in order not to include a possible run time overhead.

The script also identifies and adds contextual information about which one is the main workflow file, the workflow image, which ones are auxiliary source files of the application, and also adds `encodingFormat` contextual information when needed. All these terms included comply with the RO-Crate specification and Workflow RO-Crate profile, as mentioned earlier.

E. Resulting crate sub-directory

The result of the post-process script is a sub-directory under the application's working directory named `COMPSs_RO-Crate_[uuid]`, which is the *crate* defined in the specification. It contains: the source files of the application, the files with extra information (`command_line_parameters.txt`, `generated_graph.pdf`, `App_Profile.json`), and the RO-Crate JSON-LD file `ro-crate-metadata.json`, that describes the whole package and the application from which provenance has been recorded. Due to the impossibility of reproducing in this paper the whole JSON-LD file, an example of a COMPSs generated `ro-crate-metadata.json` is provided online[20] and also available in our use case results (see Section V). As mentioned earlier, we have ensured to be compliant with the Workflow RO-Crate profile, thus related terms to it can be found in the JSON-LD, such as: `ComputationalWorkflow`, `image`, `input`, `output` or `programmingLanguage`. The record includes a single run of a workflow.

This JSON-LD example is also useful to show that we have included the ability to automatically describe directories as input or output parameters of the workflow. COMPSs already supported this feature, but we had to enhance our post-processing to: identify sub-directories and their direction, and inspect them recursively to include the information of all their files in the `ro-crate-metadata.json`, as can be seen in the example. The sub-directory itself is included in the RO-Crate as a `Dataset` and all the files that belong to it as its `hasPart` term. It is important to mention that, while directories are internally represented in COMPSs as `dir://` URI scheme (as facilitated by the corresponding Java libraries), the RO-Crate scheme represents directories as a `file://` that ends with a slash, thus we have had to handle the translation of these URIs to ensure compliance.

[20]https://compss-doc.readthedocs.io/en/3.0/Sections/05_Tools/04_Data_Provenance.html

F. Provenance querying and reproducibility

Our current approach to handle the generated provenance is to keep it in the working directory where the workflow is run. Users with access to the specific path will be able to *grep* all results to query for provenance of past executed workflows, ensuring sensitive data is not exposed. This could be extended with the set up of a local database that stores the generated RO-Crates. In case of users willing for a public exposure of their workflow provenance, we rely on WorkflowHub, a workflow registry that is interoperable with RO-Crate.

Reproducibility of a workflow by a new user in the same supercomputer is very simple. The new user can download or copy the crate generated (i.e., the sub-directory containing the source files and RO-Crate) to their working directory in the file system, and submit the main Python application specified by the `ComputationalWorkflow` term, with the parameters specified in the `command_line_parameters.txt` file. File access permissions should be checked for inputs and outputs of the workflow, although they can be easily managed by UNIX group identifiers. The correctness of results can be checked by using diff on the output files of the workflow. If a different machine or supercomputer should be used, data assets should be made available there, as well as any software the application depends on. Finally, in all cases, replicability would require an extra effort, by changing the input files used in the workflow.

V. Use case examples

As detailed in Section IV, two of our main requirements to implement automatic gathering of workflow provenance are that it must be *scalable* and *efficient*. Scalable in the sense that our implementation must support workflows with a large number of both tasks to be executed (this is, the nodes in the graph), and input/output data assets (i.e., files). Efficient meaning that, when provenance is generated, the possible overhead added to the execution must be negligible. Thus, in order to study how good is our implementation in both scalability and efficiency terms, we have selected two real use case scenarios: Lysozyme in Water (from life sciences domain) and BackTrackBB (from earth sciences domain). Both cases have been selected due to their large number of tasks and files (read or created during execution).

Our tests have been run in the MareNostrum IV supercomputer[21], using an adequate number of computing nodes for each corresponding case depending on its workload, and having dedicated computing nodes for both the COMPSs master and each worker to avoid any interference during our measurements. Input and output files are accessed through the GPFS file system provided in the supercomputer. For each of the use cases, we have run the application 5 times, getting the average time and Confidence Intervals (CIs) of the application execution time, the graph image conversion time (from DOT to PDF, to produce a graphical image of the COMPSs generated workflow), and the generation time

[21]https://www.bsc.es/marenostrum/marenostrum/technical-information

of the resulting RO-Crate where provenance is finally stored. This way, we will understand the execution time differences when provenance is activated, and the extra time needed to generate the graph image and the RO-Crate, together with their variability.

In addition, since it is very difficult to reproduce in this paper the resulting RO-Crates, we will make them publicly available using WorkflowHub (see Section III-B). The published information includes the whole sub-directory crate generated, that includes not only the RO-Crate file, but also the source files of the application, and the rest of extra files as described in Section IV-B.

A. Lysozyme in Water

This case is a COMPSs implementation of the Lysozyme in Water example included in the GROMACS Tutorial[22], which creates a simulation system containing a set of proteins (lysozymes) in boxes of water, with ions. We have used a single computing node for the master part of the COMPSs application, and two worker computing nodes to run tasks (48 cores at each computing node). The generated workflow of this case is composed of 1336 tasks, that overall require 171 input files (43 MB size), and at the end of its execution 1503 output files are created (2.2 GB size). It is also interesting to remark that, during the execution, the COMPSs runtime has registered in the `dataprovenance.log` a total of 4175 accesses to files (either inputs, outputs or inouts), while tasks are executed. As previously explained, this is the main information the RO-Crate post-process script uses to generate the provenance.

TABLE I
LYSOZYME IN WATER EXECUTION TIMES (ALL IN SECONDS)

	Average time	Confidence Interval
No Provenance	113,6	$\pm 2,78$
Provenance	112,85	$\pm 1,54$
Graph conversion	38,1	$\pm 0,4$
RO-Crate creation	16,55	$\pm 0,38$

Table I shows the average times when executing Lysozyme in Water with and without provenance activated, together with the graph image conversion and RO-Crate creation times, and all their CIs. From the numbers, we can see that the activation of the provenance registration does not add any noticeable run time overhead to the execution of the workflow. The extra time is paid after the workflow execution ends, with the post-process that creates the graph image and the RO-Crate. The DOT to PDF graph image conversion takes quite some time in this case, due to the fact that we are dealing with a big workflow (1336 tasks). The RO-Crate creation time includes both the time of processing the `dataprovenance.log` and the use of the ro-crate-py library to write the resulting `ro-crate-metadata.json`, while also copying the files to the crate sub-directory, that are 2.45 MB big in size. We have made available the resulting files at WorkflowHub[23].

[22]http://www.mdtutorials.com/gmx/lysozyme/index.html

[23]https://doi.org/10.48546/workflowhub.workflow.379.1

B. BackTrackBB

BackTrackBB [21] is an application for the detection and space-time location of seismic sources based on multi-scale, frequency-selective statistical coherence of the wave field recorded by dense large-scale seismic networks and local antennas. It performs signal processing, space-time imaging and detection and location in order to enhance coherence of the signal statistical features across the seismic sources. This second case has been executed using 10 MareNostrum IV computing nodes, 1 for the COMPSs master, and 9 for the workers executing tasks (again, with 48 cores available per computing node). We have configured the BackTrackBB workflow to run 1 simulation day with 100 stations, which creates a workflow of 700 tasks, and reads 2400 input files (7.1 GB total size) to execute them, generating 48 final outputs (37 MB). The `dataprovenance.log` has recorded 2448 accesses to files, that were used to generate the crate sub-directory with the RO-Crate and source files (22 MB size). Again, we have uploaded in WorkflowHub[24].

TABLE II
BACKTRACKBB EXECUTION TIMES (ALL IN SECONDS)

	Average time	Confidence Interval
No Provenance	3799,65	$\pm 53,24$
Provenance	3772,05	$\pm 39,14$
Graph conversion	3,72	$\pm 0,06$
RO-Crate creation	37,02	$\pm 0,34$

Table II shows the same times gathered for Lysozyme in Water, but now for BackTrackBB: average execution time with provenance deactivated and activated, graph image conversion, and RO-Crate creation times (plus CIs). We can observe that, execution times are quite longer than in the Lysozyme case, but again we do not see an influence in the execution time when the automatic provenance registration is activated in COMPSs, since the resulting execution times are inside the Confidence Intervals. The most curious results come from the graph image conversion and RO-Crate creation times. In the former, the time is an order of magnitude smaller than in Lysozyme, which we attribute to the fact that the workflow is smaller (i.e., less task nodes, less edges). Anyway, we consider the specific time of graph image generation and how to improve it out of scope of this paper. In the latter case (RO-Crate creation time), we see that the time is more than double of the Lysozyme case. Comparing both applications, we see that BackTrackBB registers less accesses to files than the Lysozyme, therefore we cannot consider that the number of file accesses is negatively influencing the RO-Crate creation time. The larger number of input and output files used in BackTrackBB (2448 vs 1674 in Lysozyme) seems to be the main cause of increase in the RO-Crate creation time. Although this time is reasonable compared to the application's execution time, we will investigate in the future if there is room for improvement in the time taken when adding a file record in the RO-Crate using the *ro-crate-py* library.

[24]https://doi.org/10.48546/workflowhub.workflow.386.1

VI. Conclusions

This paper proposes to achieve FAIR HPC workflows through the adoption of the RO-Crate specification for provenance metadata registration and publication. Our workflow runs are made FAIR by being published in the WorkflowHub registry (Findable and Accessible), while leveraging the use of the RO-Crate specification, that makes them Interoperable and Reusable.

We have studied the related work and have not found any solution for large HPC workflows that is able to record data provenance avoiding run time overheads. Our lightweight design when recording provenance information at run time can inspire others to follow the same approach in their systems, avoiding to cause an increase to the makespan of the workflow.

The merging of COMPSs with the RO-Crate specification has been able to jointly provide the strong capabilities of both: register automatically the workflow provenance information already available at the COMPSs runtime, and record it using the simple, powerful and well-established RO-Crate specification, which provides to our solution interoperability with a number of existing systems and tools. Our COMPSs RO-Crates can be understood not only by WorkflowHub, but any other tools or systems that currently adopt RO-Crate.

We have demonstrated that we meet the requirements established in our design:

- HPC workflows are supported (Section V).
- Reproducibility and replicability, specifically for workflows (Sections IV-E and IV-F).
- Automatic provenance registration (Section IV-B).
- Scalable registration to support large workflows (Section V).
- Simple provenance representation format, compatible with complex workflows (Sections III-B, IV-A, and IV-D).
- Efficient provenance registration in terms of run time (Section V).

With the results of our experiments, we can affirm that our COMPSs automatic data provenance recording method is both scalable (it supports large workflows) and efficient (because no run time overhead is added while recording provenance). We have also identified that the graph image generation time depends very much on the workflow structure. Besides, the RO-Crate generation time is not highly influenced by the number of file accesses recorded (validating our lightweight approach), but more by the number of input/output files included in the RO-Crate metadata file. These two generation times are implemented as a post-process, which avoids having to pay the extra time at run time. We will consider to deeper study generation times in our future work, together with the adaptation to other programming languages supported by COMPSs (i.e., Java and C/C++) and explore RO-Crate interoperability with other tools (e.g., WfExS).

References

[1] M. Baker, "Reproducibility crisis," *Nature*, vol. 533, no. 26, pp. 353–66, 2016.

[2] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[3] P. Missier, K. Belhajjame, and J. Cheney, "The W3C PROV family of specifications for modelling provenance metadata," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 773–776.

[4] P. Buneman, S. Khanna, and T. Wang-Chiew, "Why and where: A characterization of data provenance," in *International conference on database theory*. Springer, 2001, pp. 316–330.

[5] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A survey on provenance: What for? What form? What from?" *The VLDB Journal*, vol. 26, no. 6, pp. 881–906, 2017.

[6] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for computational tasks: A survey," *Computing in science & engineering*, vol. 10, no. 3, pp. 11–21, 2008.

[7] B. Clifford, I. Foster, J.-S. Voeckler, M. Wilde, and Y. Zhao, "Tracking provenance in a virtual data grid," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 565–575, 2008.

[8] S. Soiland-Reyes, P. Sefton, M. Crosas, L. J. Castro, F. Coppens, J. M. Fernández, D. Garijo, B. Grüning *et al.*, "Packaging research artefacts with RO-Crate," *Data Science*, vol. 5, no. 2, pp. 97–138, 2022.

[9] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[10] J. Köster and S. Rahmann, "Snakemake—a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012.

[11] F. Z. Khan, S. Soiland-Reyes, R. O. Sinnott, A. Lonie, C. Goble, and M. R. Crusoe, "Sharing interoperable workflow provenance: A review of best practices and their practical application in CWLProv," *GigaScience*, vol. 8, no. 11, 2019.

[12] C. Goble, S. Cohen-Boulakia, S. Soiland-Reyes, D. Garijo, Y. Gil, M. R. Crusoe, K. Peters, and D. Schober, "FAIR Computational Workflows," *Data Intelligence*, vol. 2, no. 1-2, pp. 108–121, 2020.

[13] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia, "ServiceSs: an interoperable programming framework for the Cloud," *Journal of Grid Computing*, vol. 12, no. 1, pp. 67–91, 2014.

[14] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. M. Badia, J. Torres, T. Cortes, and J. Labarta, "PyCOMPSs: Parallel computational workflows in python," *The International Journal of High Performance Computing Applications*, vol. 31, no. 1, pp. 66–82, 2017.

[15] C. Ramon-Cortes, F. Lordan, J. Ejarque, and R. M. Badia, "A Programming Model for Hybrid Workflows: combining Task-based Workflows and Dataflows all-in-one," *Future Generation Computer Systems*, vol. 113, pp. 281–297, 2020.

[16] C. Ramon-Cortes, A. Serven, J. Ejarque, D. Lezzi, and R. M. Badia, "Transparent orchestration of task-based parallel applications in containers platforms," *Journal of Grid Computing*, vol. 16, no. 1, pp. 137–160, 2018.

[17] J. Ejarque, M. Bertran, J. Á. Cid-Fuentes, J. Conejero, and R. M. Badia, "Managing failures in task-based parallel workflows in distributed computing environments," in *European Conference on Parallel Processing*. Springer, 2020, pp. 411–425.

[18] P. Sefton, E. Ó Carragáin, S. Soiland-Reyes, O. Corcho, D. Garijo, R. Palma, F. Coppens, C. Goble, J. M. Fernández, K. Chard *et al.*, *RO-Crate Metadata Specification 1.1.2*, Jan. 2022, Recommendation published by researchobject.org - see https://w3id.org/ro/crate/1.1.

[19] P. De Geest, B. Droesbeke, I. Eguinoa, A. Gaignard, S. Huber, S. Leo, L. Pireddu, L. Rodríguez-Navas, R. Sirvent, and S. Soiland-Reyes, "ro-crate-py," May 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6594974

[20] C. Goble, S. Soiland-Reyes, F. Bacall, S. Owen, A. Williams, I. Eguinoa, B. Droesbeke, S. Leo, L. Pireddu *et al.*, "Implementing FAIR Digital Objects in the EOSC-Life Workflow Collaboratory," Mar. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4605654

[21] N. Poiata, C. Satriano, J.-P. Vilotte, P. Bernard, and K. Obara, "Multi-band array detection and location of seismic sources recorded by dense seismic networks," *Geophysical Journal International*, vol. 205, no. 3, pp. 1548–1573, 2016.