Automatic, Efficient and Scalable Provenance Registration for FAIR HPC Workflows

Raül Sirvent, Javier Conejero, Francesc Lordan, Jorge Ejarque, Laura Rodríguez-Navas, José M. Fernández, Salvador Capella-Gutiérrez, Rosa M. Badia

17th Workshop on Workflows in Support of Large-Scale Science
November 14, 2022. Dallas, TX, USA
Motivation

- Volume of data generated from **scientific workflow** experiments continues to grow, important to manage results
- Reproducibility crisis: provide more than just numbers on a scientific paper
- **FAIR** and **provenance** registration to achieve **reproducibility** and **replicability**
  - Visual tools difficult to scale
  - Custom formats that complicate interoperability
  - RDF and OWL learning curve for non-experts on SemanticWeb
- Our proposal for scientific workflow provenance registration
  - **Automatic**: users do not need to provide annotations on what to record
  - **Efficient**: lightweight approach to avoid run time overheads (target HPC)
  - **Scalable**: large workflows (thousands of task nodes and / or files used)
Related Work

Provenance for DBs (Buneman et al., 2001)

Early WfMS Provenance (Freire et al., 2008)
Prospective vs retrospective (DB stored, some RDF & OWL)

PROV (Missier et al., 2013)
W3C Provenance WG

FAIR (Wilkinson et al., 2016)

Nextflow (Di Tommaso et al., 2017): log command to build provenance manually

RO-Crate (Soiland-Reyes et al., 2022)

Snakemake (Köster et al., 2021): report from user’s annotations

CWLProv (Khan et al., 2019): automatic, but overhead at run time

Our approach: automatic, efficient and scalable
Background: COMPSs

- **Sequential** programming, parallel execution
- **General purpose** programming language + annotations/hints (identify tasks and directionality of data)
- Builds a task graph at runtime (potential concurrency)
- Tasks can be sequential, parallel (threaded or MPI)
- Offers to applications a shared memory illusion in a distributed system (Big Data apps support)
- Support for persistent storage
- **Agnostic** of computing platform: enabled by the runtime for clusters, clouds and container managed clusters

- **Advanced features**: heterogeneous infrastructures, task constraints, streamed data, task faults, task exceptions, checkpointing, elasticity
Background: Research Object Crate

• Evolution from:
  • Research Object: describe and link digital and real-world resources
  • DataCrate: describe and aggregate data with associated metadata

• Lightweight approach to package research data with their metadata

• Wide scope: from an individual researcher working with a folder of data, to large data-intensive computational research environments

• RO-Crate Workflow profile (narrow down)
  • Set of conventions, types and properties to allow interoperability

• Machine-readable JSON Linked Data (JSON-LD)
  • Main vocabulary based on Schema.org
  • Structure: Root Data Entity, Data Entities (files, directories), Contextual Entities (non-digital elements)

• Strong ecosystem:
  • ro-crate-py library
  • WorkflowHub
Design Requirements

- Target HPC workflows (commonly large)
- Reproducibility and replicability for workflows
- **Automatic** provenance registration
- **Scale** to large workflows (thousands of files and tasks)
- **Efficient** provenance registration (avoid significant overheads at run time)
- Provenance representation format: simple but able to represent complex workflows
COMPSs runtime modifications

- Lightweight approach: record file accesses, generate provenance later

After application finishes...

- Flags -p or --provenance trigger it after execution
- Can be manually invoked if provenance generation time becomes an issue (i.e., extreme large workflows)

3.0.rc2206
lysozyme_in_water.py
App_Profile.json
file://s01r2b54-ib0/home/bsc19/bsc19057/DP_Test_3_demo/dataset/2hs9.pdb IN
file://s01r2b54-ib0/home/bsc19/bsc19057/DP_Test_3_demo/output/2hs9.gro OUT
file://s01r2b54-ib0/home/bsc19/bsc19057/DP_Test_3_demo/output/2hs9.top OUT
...

It's the crate
- ro-crate-metadata.json
- Application source files, command line arguments, workflow image and profile
Crate Assets Included

- Data assets to be included in the *crate* (package)
  - Application source code files

```json
"@id": "matmul_files.py",
"@type": ["File", "SoftwareSourceCode", "ComputationalWorkflow"],
"contentSize": 1948,
"description": "Main file of the COMPSs workflow source files",
"encodingFormat": "text/plain",
"image": {"@id": "complete_graph.pdf"},
"input": [{"@id": "file://s01r1b56-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/A.0.0"}, ...]
"name": "matmul_files.py",
"output": [{"@id": "file://s01r1b56-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/C.0.0"}, ...]
"programmingLanguage": {"@id": "#compss"}
```

- Workflow image (PDF)

```json
"@id": "complete_graph.pdf",
"@type": ["File", "ImageObject", "WorkflowSketch"],
"about": {"@id": "matmul_files.py"},
"contentSize": 14558,
"description": "The graph diagram of the workflow, automatically generated by COMPSs runtime",
"encodingFormat": [{"application/pdf","@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/276"}],
"name": "complete_graph.pdf"
```
Crate Assets Included

- Data assets to be included in the crate (package)
  - Command line arguments

```json
"@id": "compss_command_line_arguments.txt",
"@type": "File",
"contentSize": 4,
"description": "Parameters passed as arguments to the COMPSs application through the command line",
"encodingFormat": "text/plain",
"name": "compss_command_line_arguments.txt"
```

- COMPSs application profiling (task statistics per resource used)

```json
"@id": "App_Profile.json",
"@type": "File",
"contentSize": 404,
"description": "COMPSs application Tasks profile",
"encodingFormat": ["application/json", {"@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/817"}],
"name": "App_Profile.json"
```
Crate Assets Not Included

- Assets not directly included: input, output files (or directories) of the workflow
  - Avoid big movements of data
  - Added them as URIs
    - Feedback to RO-Crate community

```
"@id": "file://s02r2b26-ib0/home/bsc19/bsc19057/DP_Test_3_demo/config/energy.selection"
```

- Automatically identified using runtime knowledge: no need for users to specify them

```
"@id": "file://s07r1b33-ib0/home/bsc19/bsc19057/DP_Test_3_demo/output/8lyz_solv_ions.gro",
"@type": "File",
"contentSize": 1643019,
"name": "8lyz_solv_ions.gro",
"sdDatePublished": "2022-10-18T08:03:08+00:00"
```
Other Information

- contentSize and sdDatePublished (modification date) to ensure files have not been altered

```
"@id": "file://s07r1b33-ib0/home/bsc19/bsc19057/DP_Test_3_demo/output/8lyz_solv_ions.gro",
"@type": "File",
"contentSize": 1643019,
"name": "8lyz_solv_ions.gro",
"sdDatePublished": "2022-10-18T08:03:08+00:00"
```

- Non-automatically gathered info: ro-crate-info.yaml

```
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/
```
Use Case: Lysozyme in Water

- GROMACS Tutorial: simulation system containing a set of **proteins** (lysozymes) in boxes of water, with ions
- MareNostrum IV: 48 cores per node

<table>
<thead>
<tr>
<th>Worker Nodes</th>
<th>2 (96 cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>1336</td>
</tr>
<tr>
<td>Input files</td>
<td>171 (43 MB)</td>
</tr>
<tr>
<td>Output files</td>
<td>1503 (2.2 GB)</td>
</tr>
<tr>
<td>dataprovenance.log</td>
<td>4175 file accesses</td>
</tr>
<tr>
<td>Resulting Crate size</td>
<td>2.45 MB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Average time</th>
<th>Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Provenance</td>
<td>113.6</td>
<td>±2, 78</td>
</tr>
<tr>
<td>Provenance</td>
<td>112.85</td>
<td>±1.54</td>
</tr>
<tr>
<td>Graph conversion</td>
<td>38.1</td>
<td>±0.4</td>
</tr>
<tr>
<td>RO-Crate creation</td>
<td>16.55</td>
<td>±0.38</td>
</tr>
</tbody>
</table>

https://doi.org/10.48546/workflowhub.workflow.379.1
Use Case: BackTrackBB

- Multi-band array detection and location of **seismic sources**
- MareNostrum IV: 48 cores per node

<table>
<thead>
<tr>
<th>Worker Nodes</th>
<th>9 (432 cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>700</td>
</tr>
<tr>
<td>Input files</td>
<td>2400 (7.1 GB)</td>
</tr>
<tr>
<td>Output files</td>
<td>48 (37 MB)</td>
</tr>
<tr>
<td>dataprovenance.log</td>
<td>2448 file accesses</td>
</tr>
<tr>
<td>Resulting Crate size</td>
<td>22 MB</td>
</tr>
</tbody>
</table>

| No Provenance      | 3799.65       | ±53.24 |
| Provenance         | 3772.05       | ±39.14 |
| Graph conversion   | 3.72          | ±0.06  |
| RO-Crate creation  | 37.02         | ±0.34  |

https://doi.org/10.48546/workflowhub.workflow.386.1
Conclusions

- **FAIR HPC workflows** combining RO-Crate + COMPSs + WorkflowHub
  - Feedback to RO-Crate community with our HPC case
- No previous solution for large HPC workflows that studies and avoids run time overheads
- Our experiments show
  - We can **scale** and deal with large workflows
  - We are **efficient** (no run time overhead appreciated)
  - We provide **automatic** provenance registration, whenever possible
  - RO-Crate generation time using ro-crate-py library
    - Not highly influenced by the number file accesses recorded (dataprovenance.log)
    - It is influenced by the number of input/output files included
  - Graph image generation time becomes an issue sometimes (out of scope)
Thanks for your attention!

This work has been supported by the Spanish Government (PID2019-107255GB-C21)

Raul.Sirvent@bsc.es