SOFTWARE FACTORIES FOR REPRODUCIBLE RESEARCH IN BIG DATA/DL/AI

Arnaud Legrand

July 2020
Publishing: the Reproducibility "Wave"

Artifact evaluation and ACM badges

Emerging Interest Group on https://reproducibility.acm.org/
Publishing: the Reproducibility "Wave"

Artifact evaluation and ACM badges

Emerging Interest Group on https://reproducibility.acm.org/

Major conferences

- ACM SIGMOD 2015-2019, ACM MM 2019-2020, ...
- Supercomputing: Artifact Description (AD) mandatory, Artifact Evaluation (AE) optional, Double blind vs. Open reviews
- NeurIPS, ICLR: open reviews, Joelle Pineau @ NeurIPS’18

Mindsets and practices are evolving: people care and make stuff available
But it’s hard and tools are not mature yet

Why focus on AI/DL? Not only but it is very active, empirical and computational field
my_code --cfg=magical_param:0.94572 '*.dat' --output foo.csv

Tracking code version

- my_code is revision 21b95ecfa0911d6ca87668482b11ab9498edd8f3
my_code --cfg=magical_param:0.94572 '*.dat' --output foo.csv

Tracking code version
  • my_code is revision 21b95ecfa0911d6ca87668482b11ab9498edd8f3

Tracking software environment
  • my_code depends on a dozen of libraries, which depend on dozens of libraries
  • my_code was compiled with clang 1:9.0-49.1 and -O3 -funroll-loops
    -fno-strict-aliasing -finline-functions...
Main Challenges

```
my_code --cfg=magical_param:0.94572 '*.dat' --output foo.csv
```

Tracking code version
- `my_code` is revision 21b95ecfa0911d6ca87668482b11ab9498edd8f3

Tracking software environment
- `my_code` depends on a dozen of libraries, which depend on dozens of libraries
- `my_code` was compiled with `clang 1:9.0-49.1` and `-O3 -funroll-loops -fno-strict-aliasing -finline-functions ...`

Tracking parameters and data
- `*.dat`? Ooh, you ran this in `data/2091293-AJXQ37`?
- Wasn’t `mymap.dat` updated since then?
- That was for `foo.csv`. What about `bar.csv`? Is it reproducible?
Main Challenges

```
my_code --cfg=magical_param:0.94572 '*.dat' --output foo.csv
```

Tracking code version

- `my_code` is revision `21b95ecfa0911d6ca87668482b11ab9498edd8f3`

Tracking software environment

- `my_code` depends on a dozen of libraries, which depend on dozens of libraries
- `my_code` was compiled with `clang 1:9.0-49.1` and `-O3 -funroll-loops -fno-strict-aliasing -finline-functions ...`

Tracking parameters and data

- `*.dat`? Ooh, you ran this in `data/2091293-AJXQ37`?
- Wasn’t `mymap.dat` updated since then?
- That was for `foo.csv`. What about `bar.csv`? Is it reproducible?

Tracking the process (on short/long term)

- Why did I run this? What did I learn from it? I remember doing this but when?
Main Challenges

my_code --cfg=magical_param:0.94572 '*.dat' --output foo.csv

Tracking code version
  • my_code is revision 21b95ecfa0911d6ca87668482b11ab9498edd8f3

Tracking software environment
  • my_code depends on a dozen of libraries, which depend on dozens of libraries
  • my_code was compiled with clang 1:9.0-49.1 and -O3 -funroll-loops -fno-strict-aliasing -finline-functions ...

Tracking parameters and data
  • *.dat? Ooh, you ran this in data/2091293-AJXQ37?
  • Wasn’t mymap.dat updated since then?
  • That was for foo.csv. What about bar.csv? Is it reproducible?

Tracking the process (on short/long term)
  • Why did I run this? What did I learn from it? I remember doing this but when?

Handle complex sequences and reuse results (leverage cloud/supercomputers)
I can’t be exhaustive here but feel free to help me getting a better view on all this by:

• Adding related projects in the pad
• Telling me about your experience on these tools if you ever tried them


Or come discuss about all this and share news on https://reproducible-research.inria.fr/
Existing Tools and Standards

Notebooks and workflows

Software environments

Sharing platforms

The Data/Code/Execution Triptych
Git has become the de facto standard and ultimate code tracking tool

- GitHub, BitBucket, GitLab, ... make it more human-friendly
- Note that none of these platforms is perennial (see Roberto’s presentation on SoftWare Heritage)
- Git tracks a single software project: ~> Git submodule 😊 or Git subtree 😞
Git has become the de facto standard and ultimate code tracking tool

- GitHub, BitBucket, GitLab, ... make it more human-friendly
- Note that none of these platforms is perennial (see Roberto’s presentation on SoftWare Heritage)
- Git tracks a single software project: ∨ Git submodule 😊 or Git subtree 😞

Why not use Git for tracking data?

- Git was not designed to handle large binary files (e.g., databases, HDF5, video)
  ∨ Git LFS (co-developed by GitHub and Atlassian around 2015) 😞 and Git Annex 😊
- Git was not designed with "privacy" or "access control lists" in mind
  ∨ Cryptography support in Git Annex 😊
Binary based QEMU/VirtualBox/VMWare/...  LXC/Docker/Singularity/CharlieCloud/...

- Easy to use (e.g., share through DockerHub) although the first installation requires a root access
- Provides a common/standard software stack for a team
- No (or little) support for image reproducibility nor software inspection/modification
Tracking Software Environment

**Binary based QEMU/VirtualBox/VMWare/...  LXC/Docker/Singularity/CharlieCloud/...**

- Easy to use (e.g., share through DockerHub) although the first installation requires a root access
- Provides a common/standard software stack for a team
- No (or little) support for image reproducibility nor software inspection/modification

**Source based Spack, Guix, Nix, ...**

- Allows to rebuild a software stack in a controlled way (sources, compiling options)
- Use caches to save compiling time
- Time machine mechanism and export to docker/...
Tracking Software Environment

Binary based QEMU/VirtualBox/VMWare/... LXC/Docker/Singularity/CharlieCloud/...

• Easy to use (e.g., share through DockerHub) although the first installation requires a root access
• Provides a common/standard software stack for a team
• No (or little) support for image reproducibility nor software inspection/modification

Package Managers pip, conda, dpkg, ...

• Anaconda is OS agnostic and popular for data science
  • Reinstalling a basic environment even after a few months can reveal impossible
• Debian commits to reproducibility: snapshots (March 2005) and reproducible builds
  • You may freeze a Dockerfile by wisely freezing sources
  • Limited flexibility

Source based Spack, Guix, Nix, ...

• Allows to rebuild a software stack in a controlled way (sources, compiling options)
• Use caches to save compiling time
• Time machine mechanism and export to docker/...
The REPL (Read–eval–print loop) vs. Notebook vs IDE debate

• In the beginning was the Mathematica (1988) and the Maple (1989) notebooks, which allow to tell a story (literate programming)
The REPL (Read–eval–print loop) vs. Notebook vs IDE debate

- In the beginning was the Mathematica (1988) and the Maple (1989) notebooks, which allow to tell a story (literate programming)
- Then IJulia, IPython, and IR merged into the Jupyter notebook 😊
  - The coolest kid on the block without IDE/Structure, Interactive collaboration/Versioning, Software Environment Control, Easy setup and use of Computing resources !!! 😊
The REPL (Read–eval–print loop) vs. Notebook vs IDE debate

- In the beginning was the Mathematica (1988) and the Maple (1989) notebooks, which allow to tell a story (literate programming)
- Then IJulia, IPython, and IR merged into the Jupyter notebook 😊
  - The coolest kid on the block without IDE/Structure, Interactive collaboration/Versioning, Software Environment Control, Easy setup and use of Computing resources !!! 😊

Now we have:
- JupyterLab, Binder, JupyterHub
  - Guix-Jupyter
- IDE: Rstudio (not just R), Emacs, VSstudio (Jupyter~backend)
- CodeOcean showroom, interactive

- CoCalc/SAGE notebooks, Kaggle, Google Colab, DeepNote
  real-time, versioning, custom environment
- fast.ai/nbdev merge conflict, module export, test
- Beaker, Count 😞 ???

Little support for computing ressources: SoS Polyglot Notebook/Workflow System
Store records in a DB (duration, OS, args), environment, file content, concurrent executions.

PyPet extends Sumatra with Parameter Sweep (Fork-Join, store trajectories in HDF5)

Courtesy of Andrew Davison (AMP Workshop on RR)

smt configure --executable=python \
  --main=main.py \
  --datapath /path/to/data

smt run new.param --label=Sgamma \
  --reason="Test a smaller gamma"

smt comment 20110713-174949 "Eureka! Nobel prize here I come"

smt tag "Figure 6"

smt repeat Sgamma

smtweb
Running simulations with **Sumatra** *(Computational Neuroscience)*

```
smt configure --executable=python \
    --main=main.py \ 
    --datapath /path/to/data

smt run new.param --label=Sgamma \ 
    --reason="Test a smaller gamma"

smt comment 20110713-174949 \ 
    "Eureka! Nobel prize here I come"

smt tag "Figure 6"

smt repeat Sgamma

smtweb
```

 Courtesy of Andrew Davison (AMP Workshop on RR)

Store records in a DB (duration, OS, args), environment, file content, concurrent executions

PyPet extends Sumatra with Parameter Sweep (Fork-Join, store trajectories in HDF5)
Tracking Data with **DataLad** (an other NeuroScience project)

Builds on `git annex` and `git submodule` plus small JSON metadata

```
datalad create myfirstrepo  # datalad clone/update
datalad save
```

- 3rd-party integration (owncloud, S3, figshare, GitHub/Lab)

```
datalad create-sibling-github
datalad export-to-figshare
```
Tracking Data with **Datalad** (an other Neuroscience project)

Builds on **git annex** and **git submodule** plus small JSON metadata

```bash
datalad create myfirstrepo # datalad clone/update
datalad save
```

- 3rd-party integration (owncloud, S3, figshare, GitHub/Lab)
  ```bash
datalad create-sibling-github
datalad export-to-figshare
```

- data set combination/linkage, handles meta-data (EXIF, ID3, …)

Nest modular datasets to create a linked hierarchy of datasets, and enable recursive operations throughout the hierarchy
Tracking Data with **Datalad** (an Other Neuroscience Project)

Builds on **git annex** and **git submodule** plus small JSON metadata

```bash
datalad create myfirstrepo  # datalad clone/update
datalad save
```

- 3rd-party integration (owncloud, S3, figshare, GitHub/Lab)

```bash
datalad create-sibling-github
datalad export-to-figshare
```

- data set combination/linkage, handles meta-data (EXIF, ID3, ...)

```bash
% datalad --output-format json search \
    bids.subject.sex:female bids.subject.age:24 bids.type:t1
```

**FAIR** identifies for datasets, file content and location

- metadata format homogenization to JSON-LD juxtapose representation of metadata plurality

```
% datalad --output-format json search \
    bids.subject.sex:female bids.subject.age:24 bids.type:t1
```

```json
{
    "path": "inputs/openneuro_ds008/sub-15/anat/sub-15_t1w.nii.gz",
    "metadata": {
        "datalad": "b9101f1e-ebc9-4bd5-a469-505baaa57387",
        "annex": "d41d8cd98f00b204e9800998e88a27e"",
        "url": "http://openneuro.s3.amazonaws.com/\_R1.1.0/ Z9",
    }
}
```
Tracking Data with **DATALAD** (an other Neuroscience project)

Builds on **git annex** and **git submodule** plus small **JSON** metadata

```
datalad create myfirstrepo  # datalad clone/update
datalad save
```

- 3rd-party integration (owncloud, S3, figshare, GitHub/Lab)

```
datalad create-sibling-github
datalad export-to-figshare
```

- Data set combination/linkage, handles meta-data (EXIF, ID3, ...)

```
datalad run -m "create a list" "bash src/list_titles.sh > data/lst.tsv"
datalad rerun eee1356bb7e8f921174e404c6df6aadcc1f158f0
```

- Basic provenance and reproducibility support

- Extension for running in containers
Data Version Control with **DVC** (Machine Learning)

- **git annex** but special `.dvc` files with information about local/remote storage

  ```bash
  dvc add data/data.xml
  git commit data/data.xml.dvc -m "Dataset updates"
  dvc push
  
  git checkout
dvc checkout # Get the content
  ```

- **snakemake** but Data Pipelines through a **dvc.yaml** task description

  ```bash
  dvc run -n prepare \
  -p prepare.seed,prepare.split \
  -d src/prepare.py -d data/data.xml \
  -o data/prepared \
  python src/prepare.py data/data.xml
  
dvc repro
  ```

- A basic Workflow Management System with little support for running pipelines in parallel
At Netflix, we’ve put substantial effort into adopting notebooks as an integrated development platform.

- Data scientist, data/reliability engineer, machine learning engineer

- Notebooks as functions with papermill (Parameterized Notebooks)
At Netflix, we’ve put substantial effort into adopting notebooks as an integrated development platform.

- Data scientist, data/reliability engineer, machine learning engineer

- Notebooks as functions with [papermill](https://papermill.readthedocs.io/) (Parameterized Notebooks)
- Run in docker environments
- **MESON**: a home made workflow orchestration/scheduler (100,000+ jobs per day)
  - Store results in S3 by default
  - Event/Time Trigger, Wait-for
  - Dashboard like usage to generate daily reports

Probably needed with so many people but I’ve no idea of how they survive such a complexity level
Super Integrated Approaches: DeepKit

- Python and Neural Network centric
- Execute, track and ML experiments
- Layer debugger for Keras2 and Pytorch
- Side by side file and metrics diff

https://deepkit.ai/assets/images/deepkit-v2020.mp4
Lord, Have mercy!

- **PolyAxon:**
  - Integrates with Keras, TensorFlow, SciKitLearn, ...
  - Tracks key model metrics, hyperparams, visualizations, artifacts and resources,
  - Version control code and data
  - Kubernetes

- **Pachyderm** (Version-controlled data science)
  - Versionned data without *git*: a centralized location (no conflict, allows removal,...)
  - Pipelines and distributed computer: Kubernetes/KubeFlow

- **Kono**

- ...
The industrialization of scientific research (Konrad Hinsen’s Blog, 2019)

The underlying cause for the reproducibility crisis is the ongoing industrialization of scientific research.

Most software was written for in-lab use, and not even made available to others. Only a small number of basic, standardized, and widely used tools, such as compilers, were already industrial products. Most data were likewise not shared outside the research group. Science is intrinsically a bottom-up process, whereas management is about top-down organization.

Towards Long-term and Archivable Reproducibility (software collapse)

Reproducible workflow solutions commonly use high-level technologies that were popular when they were created, providing an immediate solution which is unlikely to be sustainable in the long term. (Python 2 vs. Python 3)

The cost of staying up to date within this rapidly-evolving landscape is high. No dependency beyond a POSIX-compatible operating system, no administrator privileges, no network connection and storage primarily in plain text.