Design Pattern for Analysis Automation on Distributed Resources using Luigi Analysis Workflows

Marcel Rieger

CERN

CHEP 2019  05.11.2019
Motivational questions

- **Portability**: Does the analysis depend on ...
  - where it runs?
  - where it stores data?
    ▶ Execution/storage should not dictate code design!

- **Reproducibility**: When a Postdoc / PhD student leaves, ...
  - can someone else run the analysis?
  - is there a loss of information? Is a new framework required?
    ▶ Dependencies often only exist in the physicists head!

- **Preservation**: After an analysis is published ...
  - are people investing time to preserve their work?
  - can it be repeated after O(years)?
    ▶ Daily working environment should provide preservation features out-of-the-box!
Landscape of HEP analyses

- **Scale:**
  - Measure of resource consumption and amount of data to be processed
- **Complexity:**
  - Measure of granularity and inhomogeneity of workloads

- **Future analyses likely to be large and complex**
  - Undocumented structure & requirements between workloads only exist in the physicist’s brain
  - Manual execution / steering of jobs, bookkeeping of data, revisions, ...
  - Error-prone & time-consuming

→ Analysis workflow management essential for future analyses!
Example: ttbb cross section measurement
Example: ttbb cross section measurement
• Python package for building complex pipelines
• Development started at Spotify, now open-source and community-driven

Building blocks

1. Workloads defined as **Task** classes
2. Tasks **require** other tasks & output **Targets**
3. **Parameters** customize tasks and control behavior

• Web interface, error handling, command line tools, task history, collaborative features, …
• [github.com/spotify/luigi](https://github.com/spotify/luigi)
# reco.py

```python
import luigi

from analysis.ttH.tasks import Selection

class Reconstruction(luigi.Task):
    
    dataset = luigi.Parameter(default="ttH_bb")

    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return luigi.LocalTarget("reco_%s.root" % self.dataset)

    def run(self):
        inp = self.input()  # this is "output()" of Selection
        outp = self.output()

        # run the reconstruction based on "inp" to create "outp"
```

> python reco.py Reconstruction --dataset ttbar
Luigi’s execution model is make-like

1. Create dependency tree for triggered task
2. Determine tasks to actually run:
   - Walk through tree (top-down)
   - For each path, stop when all output targets of a task exist

- Only processes what is really necessary
- Error handling & automatic re-scheduling
- Scalable through simple structure
Example trees
Example trees

Work of a B.Sc. student after 2 weeks!
law - luigi analysis workflow

- **law**: layer on top of luigi (i.e. it does not replace luigi)

- Software design follows 2 primary goals:
  1. Scalability on HEP infrastructure (but not limited to)
  2. Decoupling of **run locations**, **storage locations** & **software environments**
     - No fixation on dedicated resources
     - All components interchangeable

- Provides a toolbox to follow an **analysis design pattern**
  - No constraint on language or data structures
    - Not a framework!

- Currently used by O(10) analyses at CMS, MC production, b-tag scale factor measurements, ..
1. Job submission

- Idea: submission built into tasks, **no need to write extra code**
- Currently supported job systems: HTCondor, LSF, gLite, ARC, (CRAB)
- Mandatory features such as automatic resubmission, dashboard interface, ...
- From the `htcondor_at_cern` example:

```bash
lxplus129:law_test$ law run CreateChars --version v1 --poll-interval 0.5 --workflow htcondor
INFO: [pid 30564] Worker Worker(host=lxplus129.cern.ch, username=mrieger) running
   CreateChars(branch=-1, start_branch=0, end_branch=26, version=v1)

Going to submit 26 htcondor job(s)
submitted 1/26 job(s)
submitted 26/26 job(s)
14:35:40: all: 26, pending: 26 (+26), running: 0 (+0), finished: 0 (+0), retry: 0 (+0), failed: 0 (+0)
...
14:37:10: all: 26, pending: 0 (+0), running: 26 (+26), finished: 0 (+0), retry: 0 (+0), failed: 0 (+0)
14:37:40: all: 26, pending: 0 (+0), running: 10 (-16), finished: 16 (+16), retry: 0 (+0), failed: 0 (+0)
14:38:10: all: 26, pending: 0 (+0), running: 0 (+0), finished: 26 (+10), retry: 0 (+0), failed: 0 (+0)
INFO: [pid 30564] Worker Worker(host=lxplus129.cern.ch, username=mrieger) done!
```

`lxplus129:law_test$`
Remote targets

- Idea: work with remote files as if they were local
- Remote targets built on top of GFAL2 Python bindings
  - Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - API identical to local targets
- Mandatory features such as automatic retries, local caching, ...
- Example: working with files on EOS

"FileSystem" configuration

```
# law.cfg
[wlcg_fs]
base: root://eosuser.cern.ch/eos/user/m/mrieger
```

- Base path prefixed to all paths using this "fs"
- Configurable per file operation (stat, listdir, ...)
- Protected against removal of parent directories
2. Remote targets

- Idea: work with remote files **as if they were local**
- Remote targets built on top of GFAL2 Python bindings
  - Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - API **identical** to local targets
- Mandatory features such as automatic retries, local caching, ...
- Example: working with files on EOS

Conveniently reading remote files

```python
# read a remote json file
target = law.WLCGFileTarget("/file.json", fs="wlcg_fs")

with target.open("r") as f:
    data = json.load(f)
```
2. Remote targets

- Idea: work with remote files as if they were local

- Remote targets built on top of GFAL2 Python bindings
  - Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - API identical to local targets

- Mandatory features such as automatic retries, local caching, ...

- Example: working with files on EOS

Conveniently reading remote files

```
# read a remote json file
target = law.WLCGFileTarget("/file.json", fs="wlcg_fs")

# use convenience methods for common operations
data = target.load(formatter="json")
```
2. Remote targets

- Idea: work with remote files as if they were local
- Remote targets built on top of GFAL2 Python bindings
  - Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - API identical to local targets
- Mandatory features such as automatic retries, local caching, ...
- Example: working with files on EOS

Conveniently reading remote files

```python
# same for root files with context guard
target = law.WLCGFileTarget("/file.root", fs="wlcg_fs")

with target.load(formatter="root") as tfile:
    tfile.ls()
```
2. Remote targets

- Idea: work with remote files **as if they were local**
- Remote targets built on top of GFAL2 Python bindings
  - Supports all WLCG protocols (dCache, XRootD, GridFTP, SRM, ...) + DropBox
  - API **identical** to local targets
- Mandatory features such as automatic retries, local caching, ...
- Example: working with files on EOS

Conveniently reading remote files

```python
# multiple other "formatters" available
target = law.WLCGFileTarget("/model.pb", fs="wlcg_fs")

graph = target.load(formatter="tensorflow")
session = tf.Session(graph=graph)
```
3. Environment sandboxing

- Diverging software requirements between typical workloads is a great feature / challenge / problem

- Introduce sandboxing:
  - Run entire task in different environment

- Existing sandbox implementations:
  - Sub-shell with init file
  - Docker images
  - Singularity images
# reco.py

```python
import luigi

from analysis.ttH.tasks import Selection

class Reconstruction(luigi.Task):
    dataset = luigi.Parameter(default="ttH_bb")

    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return luigi.LocalTarget("reco_%s.root" % self.dataset)

    def run(self):
        inp = self.input()  # this is "output()" of Selection
        outp = self.output()

        # run the reconstruction based on "inp" to create "outp"
```

```bash
> python reco.py Reconstruction --dataset ttbar
```
# reco.py

```python
import luigi
import law
from analysis.ttH.tasks import Selection

class Reconstruction(law.Task):
    dataset = luigi.Parameter(default="ttH_bb")

    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.LocalFileTarget("reco_%s.root" % self.dataset)

    def run(self):
        inp = self.input()  # this is "output()" of Selection
        outp = self.output()

        # run the reconstruction based on "inp" to create "outp"
```

```
> law run Reconstruction --dataset ttbar
```
```python
# reco.py
import luigi
import law
from analysis.ttH.tasks import Selection

class Reconstruction(law.Task, law.HTCondorWorkflow):
    dataset = luigi.Parameter(default="ttH_bb")

    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.LocalFileTarget("reco_%s.root" % self.dataset)

    def run(self):
        inp = self.input()  # this is "output()" of Selection
        outp = self.output()

        # run the reconstruction based on "inp" to create "outp"

> law run Reconstruction --dataset ttbar --workflow htcondor
```
# reco.py

```python
import luigi
import law
from analysis.ttH.tasks import Selection

class Reconstruction(law.Task, law.HTCondorWorkflow):
    dataset = luigi.Parameter(default="ttH_bb")

    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.WLCGFileTarget("reco_%s.root" % self.dataset, fs="eos")

    def run(self):
        inp = self.input()  # this is "output()" of Selection
        outp = self.output()

        # run the reconstruction based on "inp" to create "outp"
```

> law run Reconstruction --dataset ttbar --workflow htcondor
# reco.py

```python
import luigi
import law
from analysis.ttH.tasks import Selection

class Reconstruction(law.SandboxTask, law.HTCondorWorkflow):
    dataset = luigi.Parameter(default="ttH_bb")
    sandbox = "docker::cern/cc7-base"

    def requires(self):
        return Selection(dataset=self.dataset)

    def output(self):
        return law.WLCGFileTarget("reco_%s.root" % self.dataset, fs="eos")

    def run(self):
        inp = self.input()  # this is "output()" of Selection
        outp = self.output()

        # run the reconstruction based on "inp" to create "outp"
```

```bash
> law run Reconstruction --dataset ttbar --workflow htcondor
```
Marcel Rieger
Luigi Analysis Workflows

15 Successful applications & examples

- ttH analysis at CMS (JHEP 03 (2019) 026)
  - Large-scale: ~100 TB of storage, ~500k tasks
  - Complex: DNNs / BDTs / MEM
  - Distributed: 7 CEs, (GPU) clusters, local machines
  - Entire analysis operable by everyone at any time

- Examples applications:
  - “Hello world” example of law
  - More complex workflows
  - Integration of HTCondor (lxplus)
  - Simplistic single-top analysis with Docker Sandboxing (with CMS OpenData)
Summary

- HEP analyses likely to increase in scale and complexity
  - Resource-agnostic workflow management and automation **essential**
  - Need for toolbox providing a design pattern, **not another framework**

- **Luigi** is able to model even complex workflows
- **Law** provides convenience & scalability on HEP infrastructure

- **All** information transparently encoded in tasks, targets & dependencies
- **Full automation** of end-to-end analyses

**luigi analysis workflow**

```markdown
[GitHub Link]
github.com/riga/law
law.readthedocs.io
```
Backup
- **law** - *luigi* analysis workflow
  - Repository: [github.com/riga/law](https://github.com/riga/law)
  - Documentation: [law.readthedocs.io](https://law.readthedocs.io) (in preparation)
  - Minimal example: [github.com/riga/law/tree/master/examples/loremipsum](https://github.com/riga/law/tree/master/examples/loremipsum)
  - HTCondor example: [github.com/riga/law/tree/master/examples/htcondor_at_cern](https://github.com/riga/law/tree/master/examples/htcondor_at_cern)
  - Contact: Marcel Rieger

- **luigi** - Powerful Python pipelining package (by Spotify)
  - Repository: [github.com/spotify/luigi](https://github.com/spotify/luigi)
  - Documentation: [luigi.readthedocs.io](https://luigi.readthedocs.io)
  - “Hello world!”: [github.com/spotify/luigi/blob/master/examples/hello_world.py](https://github.com/spotify/luigi/blob/master/examples/hello_world.py)

- Technologies
  - Docker: [docker.com](https://docker.com)
  - Singularity: [singularity.lbl.gov](https://singularity.lbl.gov)
order: structure external HEP data

- Pythonic class collection to order “soft”, external HEP data
  - physics processes & cross sections
  - campaigns & datasets
  - channels & categories
  - variables & systematics

- Some data could be centrally managed, some is analysis specific

- Run the example:

- Use as data backend:

```python
dataset_ttH = Dataset("ttH125_bb", 100,
    keys = "/ttHTobb_M125_TuneCUETP8M2.../...MINIAODSIM",
    nFiles = 119,
    nEvents = 3845992)

process_ttH125 = Process("ttH125", 100,
    label = r"St\bar{t}H$",
    xsecs = { 13: Number(0.5071, (0.058, 0.092)) })

dataset_ttH.add_process(process_ttH125)
```
Thoughts on HEP analyses

- **What is a framework?**
  - Bash scripts, python tools, crab configs, CMSSW modules, magic
  - Connections mostly exist in the physicists head

- **Documentation?**
  - Not the most beloved hobby in the physics community

- **When a M.Sc. / PhD / Postdoc leaves ...**
  - Can someone else run the analysis?
  - Is this information lost? Is a new framework required?

- **Does execution dictate code design?**
  - Does the analysis depend on where it runs?

- **From my experience: ⅔ of time required for technicalities, ⅓ for physics**
  - Physics output doubled if it was the other way round?
Abstraction: analysis workflows

- Workflow, decomposable into particular workloads
- Workloads related to each other by common interface
  - In/outputs define directed acyclic graph (DAG)
- Alter default behavior via parameters
- Computing resources
  - Run location (CPU, GPU, WLCG, ...)
  - Storage location (local, dCache, EOS, ...)
- Software environment
- Collaborative development and processing
- Reproducible intermediate and final results

→ Reads like a checklist for analysis workflow management
Tailored systems

- Structure known in advance
- Workflows static & recurring
- One-dimensional design
- Special infrastructures
- Homogeneous software requirements

→ Requirements for HEP analyses mostly orthogonal
Existing WMS: MC production

Tailored systems
- Structure known in advance
- Workflows static & recurring
- One-dimensional design
- Special infrastructures
- Homogeneous software requirements

Wishlist for end-user analyses
- Structure “iterative”, a-priori unknown
- Dynamic workflows, fast R&D cycles
- Tree design, arbitrary dependencies
- Incorporate existing infrastructure
- Use custom software, everywhere

→ Requirements for HEP analyses mostly orthogonal
<table>
<thead>
<tr>
<th>Development Process</th>
<th>Existing WMS e.g. MC Management</th>
<th>Generic Analysis WMS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>final objective known in advance</td>
<td>iterative, final composition a priori unknown</td>
</tr>
<tr>
<td>Workflow Structure</td>
<td>chain structure, mostly one-dimensional</td>
<td>tree structure, arbitrarily branched</td>
</tr>
<tr>
<td>Evolution</td>
<td>static over time, recurrent execution</td>
<td>dynamic, fast R&amp;D cycles</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>specially tailored, e.g. storage systems, DBs</td>
<td>incorporate existing, quickly adapt to changes</td>
</tr>
<tr>
<td>Applicability</td>
<td>tuned to particular use case</td>
<td>flexible, able to model every possible workflow</td>
</tr>
</tbody>
</table>

→ Existing WMS highly specialized for designated use case
→ Requirements for HEP analyses mostly orthogonal
24 Achievements

1. Toolbox providing building blocks for analyses
   → Design pattern, **not a framework** (no constraint on language or data structure)
   → Full decoupling of run locations, storage locations and software environments

2. **All** information transparently encoded in tasks, targets & dependencies
   → Results **reproducible** by developer, groups, collaboration, ...
   → Analysis preservation out-of-the-box

3. make-like execution across distributed resources
   → Reduces overhead of manual management
   → Improves cycle times & error-proneness

→ Changed paradigm from executing to defining an analysis
  → Move focus back to physics
A typical example: ML workflow with uncertainties

Nominal MC

Reconstruction

MVA Split

MVA Training → MVA Evaluation

inference

weights

train test evaluate

...
A typical example: ML workflow with uncertainties

Data

- Reconstruction
- MVA Split
- MVA Training
- MVA Evaluation
- Inference

Flow:

- Data
- Reconstruction
- MVA Split
- MVA Training
- MVA Evaluation
- Inference

Flows:

- train
- test
- evaluate
- weights

Real data flow:

- ...
MC, Syst. 1

Reconstruction → MVA Split

MVA Evaluation

Inference

MC with systematic derived from nominal sample

weights

test train

evaluate

...
A typical example: ML workflow with uncertainties

MC, Syst. II

Reconstruction

MVA Split

MVA Training

MVA Evaluation

Inference

...
The diagram illustrates the architecture of Luigi, a workflow management system.

- **Central Scheduler**
  - Registers tasks
  - Checks for next task

- **Task Tree (Workers)**
  - Load dependencies

- **Input / Output Targets**
  - Read
  - Write

- **Workers**
  - Submit as job
  - Poll status

- **Software & Images**
  - Load

- **Network**
  - Load

- **Local**
  - Register Tasks
  - Next task?

- **Remote**
  - Command-line Interface

- **User**
Scenario A: file not cached yet

Remote storage (e.g. eos)

Remote request

Local request

1. Need to access file “a.root” (has unique, path-dep. hash X)
2. Stat file “a.root”
3. File “a.root” with hash X in cache with latest mtime? → no
4. Download “a.root”
5. Store “a.root” using hash X
6. Change mtime of file to value from stat (see 2)
7. Return local path in cache
8. Work with local file

Local cache

Local machine

law/python process

Remote
Scenario B: file *already* cached

Remote storage (e.g. eos)

1. Need to access file "a.root" (has unique, path-dep. hash X)
2. Stat file "a.root"
3. File "a.root" with hash X in cache with latest mtime? → yes
4. Return local path in cache
5. Work with local file

Local machine

law/python process

Local cache

Remote

Remote request

Local request
Marcel Rieger
Luigi Analysis Workflows

> check status of `ttH-bb-semi.Selection` (taskId=EMPTY_STRING, 
  - check `DCacheFileTarget(path=/analyses/ttH_bb_semi/Select
    -> absent
  - check `DCacheFileTarget(path=/analyses/ttH_bb_semi/Select
    -> absent
  - check `SiblingTargetCollection(len=1, threshold=1.0, 0x7f
    -> absent (0/1)

> check status of `common.CreatePjlioFiles` (taskId=EMPTY_STRING, 
  - check `DCacheFileTarget(path=/analyses/ttH_bb_semi/C
    -> absent
  - check `DCacheFileTarget(path=/analyses/ttH_bb_semi/C
    -> absent
  - check `SiblingTargetCollection(len=1, threshold=1.0, 1
    -> existent (1/1)

> check status of `common.GetDatasetLFNs` (taskId=EMPTY_STRING, 
  - check `DCacheFileTarget(path=/analyses/ttH_bb_semi/
    -> existent

> check status of `common.DownloadSetupFiles` (taskId=EM
  - check `SiblingTargetCollection(len=7, threshold=1
    -> existent (7/7)

> check status of `common.UploadRepo` (dCache=marcelDESY, 
  - check `SiblingTargetCollection(len=10, threshold=
    -> absent (0/10)

> check status of `common.BundleRepo` (taskId=EMPTY_STRING, 
  - check `LocalFileTarget(path=/user/public/anal
    -> absent

> check status of `common.UploadSoftware` (dCache=marcelDESY
  - check `SiblingTargetCollection(len=10, threshold=1
    -> absent (0/10)