Distributed make-like Analyses on the Grid based on Spotify's Pipelining Package

Marcel Rieger
Motivation & background

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

- **Reproducibility**: When a M.Sc. / PhD / Postdoc 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!

- Our background: ttH, ttbb, HH analyses at CMS
  - Large & complex analysis workflows
  - Multiple MVA techniques (DNNs, BDTs, Matrix element method)
Example: ttbb cross section measurement
Example: ttbb cross section measurement
Abstraction: analysis workflows

- Workflow, decomposable into particular workloads
- Workloads related to each other by common interface
  - In/outputs define directed data flow
- 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
A typical example: deep learning workflow
A typical example: deep learning workflow

- Reconstruction
- MVA Split
  - train
  - test
  - weights
- MVA Training
- MVA Evaluation
  - evaluate
  - weights
- Inference
- Real data

...
A typical example: deep learning workflow

- Reconstruction
- MVA Split
  - train
  - test
  - MC with systematic derived from nominal sample
  - weights
- MVA Evaluation
  - evaluate
- Inference
- ...

Weights
A typical example: deep learning workflow

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

MC with systematic generated from new events

train -> test -> evaluate
weights
- 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](http://github.com/spotify/luigi)
# reco.py

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 the "output()" of Selection

        # do whatever a reconstruction does

> python reco.py Reconstruction --dataset ttJets
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
- Clear & scalable through simple structure
Example trees
Example trees

Work of a B.Sc. student after 2 weeks!
- **law**: layer **on top** of **luigi** (i.e. it does not replace **luigi**)

- Development follows 2 main 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!
11 law features (1)

1. Job submission
   ■ Idea: submission built into tasks, **no need to write extra code**
   ■ Currently supported job systems: HTCondor, LSF, gLite, ARC
     ▾ Backend not hard-coded, selectable at runtime
   ■ Mandatory features
     ▾ Automatic resubmission, dashboard interface

2. Remote targets
   ■ Idea: work with remote files **as if they were local**
   ■ Build on top of GFAL2 Python bindings
     ▾ Supports all WLCG protocols (dCache, EOS, XRootD, CERNBox, ...) + Dropbox
   ■ Mandatory features
     ▾ Automatic retries, local caching

```python
target = DCacheTarget("/path/to/file.txt")

with target.open("w") as f:
    f.write("some result")
```
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 the "output()" of Selection

        # do whatever a reconstruction does
```

```bash
> python reco.py Reconstruction --dataset ttJets
```
# 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 the "output()" of Selection

        # do whatever a reconstruction does
```

> law run Reconstruction --dataset ttJets
## law in action

```python
# reco.py

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

class Reconstruction(law.Task, law.GLiteWorkflow):
    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 the "output()" of Selection

        # do whatever a reconstruction does

> law run Reconstruction --dataset ttJets --workflow glite
```
# reco.py

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

class Reconstruction(law.Task, law.GLiteWorkflow):
    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)

    def run(self):
        inp = self.input()  # this is the "output()" of Selection
                           # do whatever a reconstruction does
```

```bash
> law run Reconstruction --dataset ttJets --workflow glite
```
# reco.py

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

class Reconstruction(law.Task, law.GLiteWorkflow):
    dataset = luigi.Parameter(default="ttH_bb")
    sandbox = "docker::rootproject/root-ubuntu16"

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

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

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

        # do whatever a reconstruction does
```

```bash
> law run Reconstruction --dataset ttJets --workflow glite
```
● HEP analyses likely to increase in scale and complexity
  ■ Analysis workflow management **essential** for success of future measurements
  ■ Need for toolbox providing a design pattern, **not a framework**

● Luigi is able to model even complex workflows
● Law adds convenience & scalability in the HEP context

● **All** information transparently encoded in tasks, targets & dependencies
● Analysis preservation out-of-the-box

● [github.com/riga/law](https://github.com/riga/law), [law.readthedocs.io](http://law.readthedocs.io)
Backup
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?
Landscape of HEP analyses

- **Scale**: measure of resource consumption and amount of data
- **Complexity**: measure of granularity and inhomogeneity of workloads

Future analyses likely to be large and complex, bottlenecks:
- Undocumented structure & requirements between workloads, only exists in the physicist’s head
- Bookkeeping of data, versions, ...
- Manual execution/steering of jobs
- Error-prone & time-consuming

→ Analysis workflow management essential for future measurements!
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
order: structure external HEP data

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

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

- More info in the intro.ipynb notebook

- Use as data backend:

```python
> law run Selection --dataset ttH125_bb --...
```
1. Toolbox providing building blocks for analyses
   → Design pattern, **not a framework** (no constraint on language or data structure)
   → Full decoupling of run location, storage location and software environment

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
Successful application: ttH analysis at CMS

- Large-scale:
  - ~80 TB of storage, ~500k tasks

- Complex:
  - DNNs/BDTs/MEM, ~70 systematic variations

- Run locations:
  - 7 CEs, local machines, GPU machines

- Storage locations:
  - 2 SEs (dCache), local disk, Dropbox, CERNBox

- Clear allocation of duties in group

- Entire analysis operable by everyone
Luigi/LAW Architecture

Central Scheduler

Task Tree (Workers)

Analysis & Task Classes

Load dependencies

Input / Output Targets

Read

Write

Workers

Submit as job

Poll status

Software & Images

Load

User

Register Tasks

Next task?

Command-line Interface

Network

Local

Remote
Scenario A: file not cached yet

Remote storage (e.g. eos)

Remote request

Local request

Local machine

Remote

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

Local cache

PWD

/tmp

law/python process

Remote request

Local request
Scenario B: file *already* cached

Remote storage (e.g. eos)

Remote request

Local request

Local machine

Remote

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

$law/python$ process

Local cache

PWD

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

check status of `common.CreatePxlioFiles` (taskId=EMPTY_STRING, tenantId=EMPTY_STRING)
  - check `DCacheFileTarget(path=/analyses/ttH_bb_semi/CreatePxlioFiles)`
    -> absent
  - check `DCacheFileTarget(path=/analyses/ttH_bb_semi/CreatePxlioFiles)`
    -> absent
  - check `SiblingTargetCollection(len=1, threshold=1.0, 0x7f)`
    -> existent (1/1)

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

check status of `common.DownloadSetupFiles` (taskId=EMPTY_STRING, tenantId=EMPTY_STRING)
  - check `SiblingTargetCollection(len=7, threshold=1.0)`
    -> existent (7/7)

check status of `common.UploadRepo` (dCache=marcelDESY, tenantId=EMPTY_STRING)
  - check `SiblingTargetCollection(len=10, threshold=1.0)`
    -> absent (0/10)

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

check status of `common.UploadSoftware` (dCache=marcelDESY, tenantId=EMPTY_STRING)
  - check `SiblingTargetCollection(len=10, threshold=1.0)`
    -> absent (0/10)
Status of projects

- Completeness: 90%
- Missing: documentation, unit tests

- Completeness: 95%
- Missing: datacards and luminosity helpers
- Optional: centrally managed processes/campaigns/datasets

- Completeness: 100%
- (scientific numbers w/ uncertainties & gaussian propagation)