Pylightnix

Sergey Mironov

February 2021

Abstract

Pylightnix is a minimalistic Python library for managing filesystem-based immutable data objects, inspired by Purely Functional Software Deployment Model thesis by Eelco Dolstra and the Nix package manager.

The library may be thought as of low-level API for creating caching wrappers for a subset of Python functions. In particular, Pylightnix allows user to

- Prepare (or, in our terms, **instantiate**) the computation plan aimed at creating a tree of linked immutable stage objects, stored in the filesystem.
- Implement (**realize**) the prepared computation plan, access the resulting artifacts. Pylightnix is able to handle possible **non-deterministic** results of the computation. As one example, it is possible to define a stage depending on best top-10 instances (in a user-defined sence) of prerequisite stages.
- Handle changes in the computation plan, re-use the existing artifacts whenever possible.
- Gain full control over all aspects of the cached data including the garbage-collection.

1 Quick start

We illustrate Pylightnix principles by showing how to use it for making a mock data science task. We'll follow the SciPy anneal example but imagine that we want to save some intermediate results and also that there are several people working on this task.

We start by importing the required Python modules.

```
from numpy import array, save, load, exp
from scipy.optimize import dual_annealing
import matplotlib.pyplot as plt

from pylightnix import *
```

1.1 The problem

The annealing example begins with defining a complex parametric function `f` for what we need to find a minimum point as close as possible.

```
def f(z, *params):
    x, y = z
    a, b, c, d, e, f, g, h, i, j, k, l, scale = params
    return (a * x**2 + b * x * y + c * y**2 + d*x + e*y + f) + \
           (-g*exp(-((x-h)**2 + (y-i)**2) / scale)) + \
           (-j*exp(-((x-k)**2 + (y-l)**2) / scale))
```

1.2 Stages basics

Lets assume that we are to get the `f`'s parameters from elsewhere and that we want to get the following results: (a) the starting parameters themselves (b) the annealing result and statistics and (c) visual plot of the results.

1.2.1 Parameter stage

```
def stage_params(m:Manager)->DRef:
  def _config():
    name = 'params'
    out = [selfref, "params.npy"]
    return locals()
  def _make(b:Build):
    save(mklens(b).out.syspath, (2, 3, 7, 8, 9, 10, 44, -1, 2, 26, 1, -2,
    ↪  0.5))
  return mkdrv(m, mkconfig(_config()), match_only(), build_wrapper(_make))
```

1.2.2 Annealing stage

```
def stage_anneal(m:Manager, ref_params:DRef)->DRef:
  def _config():
    name = 'anneal2'
```

```
    nonlocal ref_params
    trace_xs = [selfref, 'tracex.npy']
    trace_fs = [selfref, 'tracef.npy']
    out = [selfref, 'result.npy']
    return locals()
  def _make(b:Build):
    params = load(mklens(b).ref_params.out.syspath)
    xs = []; fs = []
    def _trace(x,f,ctx):
      nonlocal xs,fs
      xs.append(x.tolist())
      fs.append(f)
    res = dual_annealing(f, [[-10,10],[-10,10]],
                         x0=[2.,2.],args=params,
                         maxiter=500, callback=_trace)
    save(mklens(b).trace_xs.syspath, array(xs))
    save(mklens(b).trace_fs.syspath, array(fs))
    save(mklens(b).out.syspath, res['x'])
  return mkdrv(m, mkconfig(_config()), match_only(), build_wrapper(_make))
```

1.2.3 Plotting stage

```
def stage_plot(m:Manager, ref_anneal:DRef)->DRef:
  def _config():
    name = 'plot'
    nonlocal ref_anneal
    out = [selfref, 'plot.png']
    return locals()
  def _make(b:Build):
    xs=load(mklens(b).ref_anneal.trace_xs.syspath)
    fs=load(mklens(b).ref_anneal.trace_fs.syspath)
    res=load(mklens(b).ref_anneal.out.syspath)
    plt.figure()
    plt.title(f"Min {fs[-1]}, found at {res}")
    plt.plot(range(len(fs)),fs)
    plt.grid(True)
    plt.savefig(mklens(b).out.syspath)
  return mkdrv(m, mkconfig(_config()), match_latest(),
  ↪  build_wrapper(_make))
```

1.2.4 Running the experiment

```
ds=instantiate_inplace(stage_params)
cl=instantiate_inplace(stage_anneal,ds)
vis=instantiate_inplace(stage_plot,cl)
rref=realize_inplace(vis)
print(rref)
```

1.3 Nesting stages

```
def stage_all(m:Manager):
  ds=stage_params(m)
  cl=stage_anneal(m,ds)
  vis=stage_plot(m,cl)
  return vis
```

```
rref=realize(instantiate(stage_all))
print(rref)
```