data_loading

April 19, 2024

1 Data loading Quickstart

by Fabiano Lever

1.1 Import statements and configuration

For this tutorial, we will use the 'magic' configuration mode of the fab library. It autodiscrovers the config file and sets up a job on the cluster for us. You can fully customize this process if you need (look at the docs for more info).

[1]:
```
from fab.magic import config, beamtime, ursa
```

```
fab:INFO: Loading config from
/asap3/flash/gpfs/fl24/2023/data/11017906/shared/fab_config.toml
fab.maxwell:INFO: Maxwell submission node not detected, configuring local dask
distributed scheduler
```

```
<Client: 'tcp://131.169.183.131:33897' processes=96 threads=96>
```

We have a some logging messages. You can choose how many you want to see by setting the `logging_level` to `DEBUG`, `INFO` or `WARNING` in the config.

1.2 Loading a DAQ run

The `from fab.magic import ursa` statement loads the ursa instruments as defined in the config.

We can now load some data. We can specify which DAQ run we wish to load, either a single number or any iterable of run numbers. For examples, you could do `ursa.load(range(43861, None))` to load all runs after 43861. If no arguments are passed, all availabe data is loaded. This might take a couple of minutes for large beamtimes.

[2]:
```
run = ursa.load(daq_run=[43867])
```

Let's look at what we loaded

[3]:
```
run
```

[3]:
```
<xarray.Dataset>
Dimensions:     (train_id: 1693, shot_id: 90, eTof_trace: 3000)
Coordinates:
  * train_id    (train_id) uint32 1603155046 1603155047 … 1603156738
```

```
    daq_run     (train_id) float64 4.387e+04 4.387e+04 … 4.387e+04 4.387e+04
  * shot_id     (shot_id) int64 0 1 2 3 4 5 6 7 8 … 81 82 83 84 85 86 87 88 89
  * eTof_trace  (eTof_trace) float64 0.153 0.1535 0.154 … 1.651 1.652 1.653
Data variables:
    delay_set   (train_id) float32 nan nan nan nan … -450.2 -450.2 -450.2
    delay_enc   (train_id) float32 nan nan nan … 3.827e+06 3.827e+06 3.827e+06
    uv_diode    (train_id, shot_id) float32 dask.array<chunksize=(565, 90),
meta=np.ndarray>
    eTof        (train_id, shot_id, eTof_trace) int16 dask.array<chunksize=(565,
90, 3000), meta=np.ndarray>
    BAM         (train_id, shot_id) float32 dask.array<chunksize=(565, 90),
meta=np.ndarray>
    undulator   (train_id) float32 nan nan nan nan nan … 34.0 34.0 34.0 34.0
    timing      (train_id) datetime64[ns] 2023-01-30T11:05:17 … 2023-01-30T…
    GMD         (train_id, shot_id) float32 dask.array<chunksize=(565, 90),
meta=np.ndarray>
    retarder    (train_id) float32 nan nan nan nan … -5.01 -5.01 -5.01 -5.01
```

We loaded a few datasources, and for each of them we have the data indexed by `train_id`, that is the macropulse id number. Sources like `GMD` or `eTof` that have shot resolved data, also have a `shot_id` dimension. By clicking on the icon at the right of each array, we can get more information about it. Like the memory requirement and the number of `chunks` the data is divided into.

Note that loading was fairly fast. This is because the we didn't really load anything yet… All operations are `lazy`. That means that we don't actually do anything computationally intensive until we absolutely have to. Only then the data is actually loaded from disk (each `chunk` get loaded in it's own thread in parallel, to speed up the computation). This lazy behaviour is indicated by the fact that the arrays are stored as `dask.array` instead of a normal `np.ndarray`.

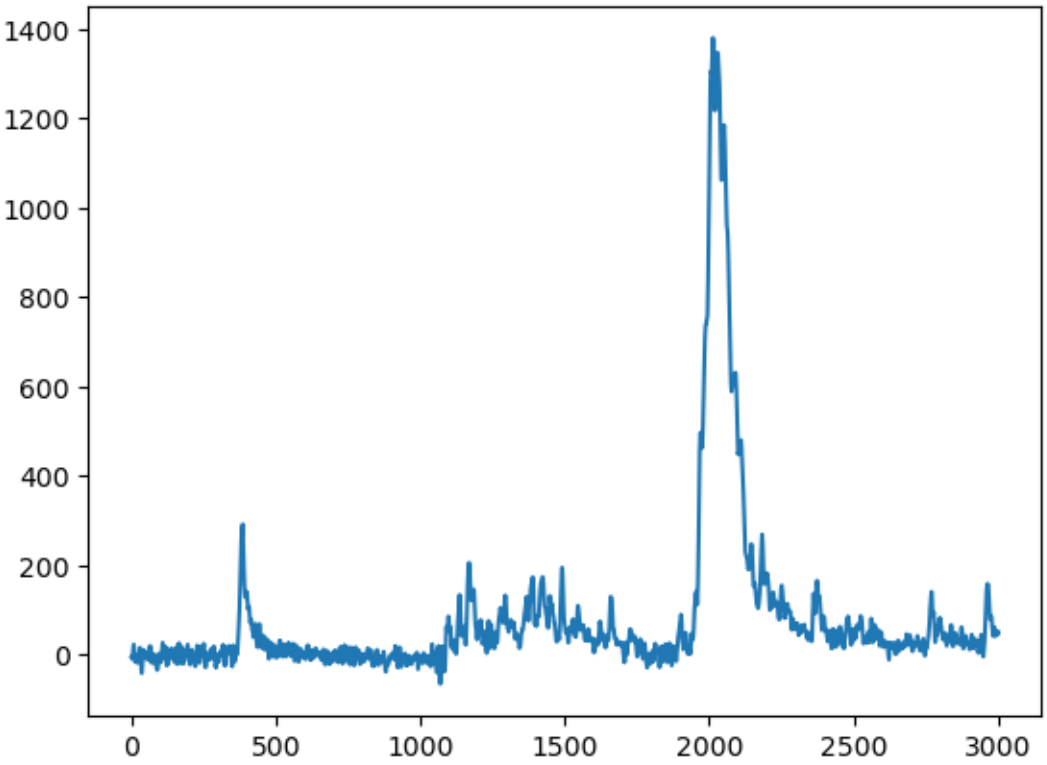
1.3 Actually loading data

Plotting something implicitly forces dask to actually load the data. Depending on how much data we want to load, this might take a while.

[4]:
```
import matplotlib.pyplot as plt

plt.plot(run.eTof[1234,0])
```

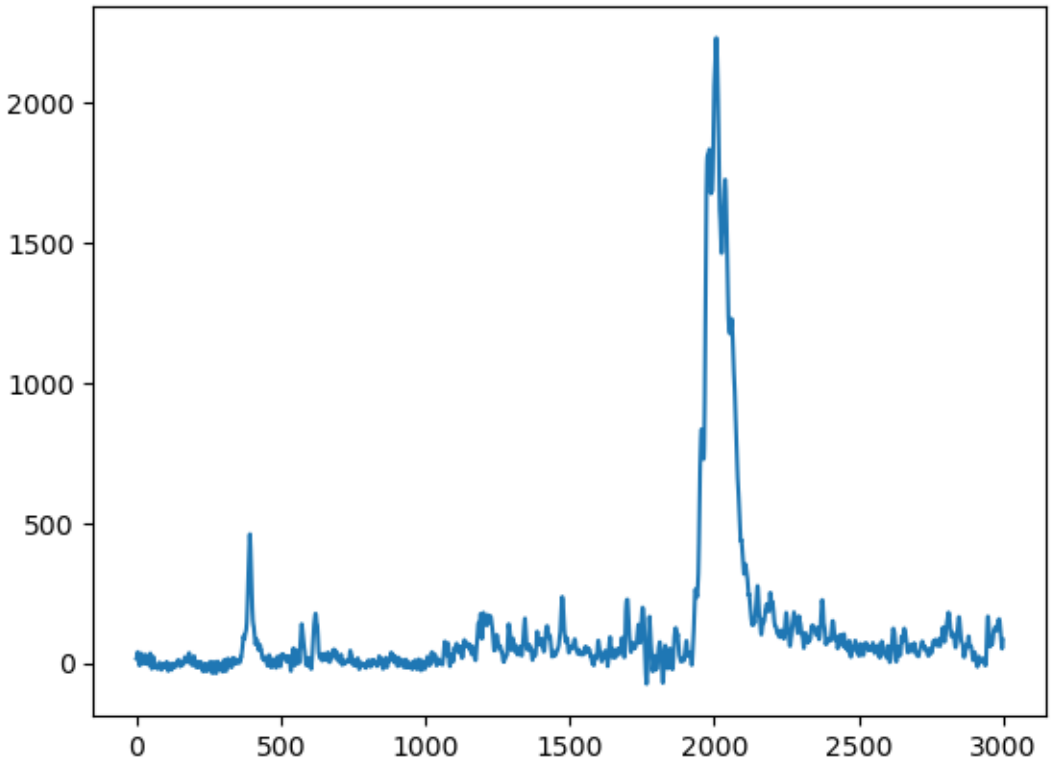
[4]:
```
[<matplotlib.lines.Line2D at 0x2b150cd13550>]
```



We can also index the data by macropulse explicitely:

[5]:
```
# .sel() selects the data by their cordinates
plt.plot(run.eTof.sel(train_id=1603156736, shot_id=44))
```

[5]:
```
[<matplotlib.lines.Line2D at 0x2b14b74d20d0>]
```



2 Yes, this is all nice, but i like my numpy arrays. How do i get one?

[10]:
```
# Just referring to the data gives us a lazy object
# You can even pass a slice object to .sel() to get a range of data
run.eTof.sel(train_id=slice(1603156726, 1603156736))
```

[10]:
```
<xarray.DataArray 'eTof' (train_id: 11, shot_id: 90, eTof_trace: 3000)>
dask.array<getitem, shape=(11, 90, 3000), dtype=int16, chunksize=(11, 90, 3000),
chunktype=numpy.ndarray>
Coordinates:
  * train_id    (train_id) uint32 1603156726 1603156727 … 1603156736
    daq_run     (train_id) float64 4.387e+04 4.387e+04 … 4.387e+04 4.387e+04
  * shot_id     (shot_id) int64 0 1 2 3 4 5 6 7 8 … 81 82 83 84 85 86 87 88 89
  * eTof_trace  (eTof_trace) float64 0.153 0.1535 0.154 … 1.651 1.652 1.653
```

[11]:
```
# If we call .compute() on the data, it will be loaded into memory
# You can even pass a slice object to .sel() to get a range of data
loaded = run.eTof.sel(train_id=slice(1603156726, 1603156736)).compute()
loaded
```

```
[11]: <xarray.DataArray 'eTof' (train_id: 11, shot_id: 90, eTof_trace: 3000)>
      array([[[-3.05500e+01, -3.35500e+01, -2.85500e+01, …, -1.55000e+00,
               -1.55000e+00,  2.45000e+00],
              [ 3.15000e+00,  1.11500e+01,  1.51500e+01, …,  1.41500e+01,
                2.41500e+01,  2.71500e+01],
              [-2.11900e+01, -2.11900e+01, -2.61900e+01, …,  5.68100e+01,
                5.98100e+01,  5.68100e+01],
              …,
              [-9.25000e-01,  1.07500e+00,  7.07500e+00, …,  4.70750e+01,
                5.00750e+01,  4.90750e+01],
              [ 8.00000e+00,  1.00000e+01,  7.00000e+00, …,  4.60000e+01,
                3.50000e+01,  3.10000e+01],
              [ 3.22250e+01,  2.72250e+01,  2.82250e+01, …,  9.42250e+01,
                7.72250e+01,  6.12250e+01]],

             [[ 2.18350e+01,  1.48350e+01,  8.83500e+00, …,  4.78350e+01,
                5.68350e+01,  6.28350e+01],
              [-6.85000e+00, -1.48500e+01, -1.48500e+01, …,  1.21150e+02,
                1.37150e+02,  1.49150e+02],
              [-2.27850e+01, -2.07850e+01, -2.07850e+01, …,  5.22150e+01,
                6.22150e+01,  6.72150e+01],
      …
              [ 1.00750e+01,  1.00750e+01,  9.07500e+00, …,  4.60750e+01,
                5.20750e+01,  5.60750e+01],
              [-9.39500e+00, -1.73950e+01, -1.53950e+01, …,  1.46050e+01,
                1.66050e+01,  1.56050e+01],
              [-1.98500e+00, -4.98500e+00, -9.98500e+00, …,  3.20150e+01,
                3.10150e+01,  3.00150e+01]],

             [[-3.27000e+00, -1.92700e+01, -1.82700e+01, …,  4.17300e+01,
                4.67300e+01,  4.57300e+01],
              [-8.06500e+00, -1.30650e+01, -1.60650e+01, …,  8.89350e+01,
                8.19350e+01,  7.29350e+01],
              [ 1.38500e+00,  9.38500e+00,  1.53850e+01, …,  6.53850e+01,
                6.63850e+01,  7.13850e+01],
              …,
              [ 8.80500e+00,  2.80500e+00, -1.19500e+00, …,  2.08050e+01,
                1.58050e+01,  1.48050e+01],
              [-1.47350e+01, -1.67350e+01, -1.67350e+01, …,  1.32650e+01,
                2.32650e+01,  2.72650e+01],
              [-7.25000e+00, -2.25000e+00, -2.50000e-01, …,  6.07500e+01,
                6.27500e+01,  6.97500e+01]]])
      Coordinates:
        * train_id    (train_id) uint32 1603156726 1603156727 … 1603156736
          daq_run     (train_id) float64 4.387e+04 4.387e+04 … 4.387e+04 4.387e+04
        * shot_id     (shot_id) int64 0 1 2 3 4 5 6 7 8 … 81 82 83 84 85 86 87 88 89
        * eTof_trace  (eTof_trace) float64 0.153 0.1535 0.154 … 1.651 1.652 1.653
```

"You promised numpy, this is still one of those newfangled xarray objects"

[12]:
```
# Ok, fine, here  is a numpy array:
# But just one shot to keep it small
run.eTof.sel(train_id=1603156726, shot_id=44).to_numpy()
```

[12]:
```
array([-2.8,  8.2,  9.2, …, 38.2, 38.2, 33.2])
```

2.1 Note on preloaded sources

Some sources (depending on their size) are automatically preloaded. In this case, there is no need to call .compute() to load them into memory. The "retarder" source, for example, is small enough to be preloaded:

[13]:
```
#It's already in memory, no need to compute
run.retarder.sel(train_id=slice(1603156726, 1603156736))
```

[13]:
```
<xarray.DataArray 'retarder' (train_id: 11)>
array([-5.01, -5.01, -5.01, -5.01, -5.01, -5.01, -5.01, -5.01, -5.01,
       -5.01, -5.01], dtype=float32)
Coordinates:
  * train_id  (train_id) uint32 1603156726 1603156727 … 1603156735 1603156736
    daq_run   (train_id) float64 4.387e+04 4.387e+04 … 4.387e+04 4.387e+04
```

[ ]:
```

```