Introduction and examples on HDF5

HDF5 is file format for storing data with many desired features, such as:

- a hierarchical structure (similar to folders/files),
- compression,
- selective data loading,
- it provides metadata features,
- fast I/O,
- it runs in multiple platforms.

In [1]:
```
import numpy as np
import tables as tb
import time
```

Saving data into an HDF5 file

Creating the file

Parameters when opening/creating and HDF5 file:

- `mode` : read, write, append. **Caveat**: using write deletes any existing file with the same name.
- `filters` : I/O filters applicable to the leaves. Not compulsory, but useful (compression).

Reference:

- https://www.pytables.org/usersguide/libref/top_level.html?highlight=open_file#tables.open_file (https://www.pytables.org/usersguide/libref/top_level.html?highlight=open_file#tables.open_file)
- https://www.pytables.org/usersguide/libref/helper_classes.html#tables.Filters (https://www.pytables.org/usersguide/libref/helper_classes.html#tables.Filters)

In [2]:
```
fpath = './test.h5'
filters = tb.Filters(complib='blosc:lz4hc', complevel=9)

with tb.open_file(fpath, mode='a', filters=filters) as h5_file:
    print(h5_file)
```

```
./test.h5 (File) ''
Last modif.: 'Wed Sep  9 18:49:48 2020'
Object Tree:
/ (RootGroup) ''
```

We have just created an empty HDF5 file with just the root group.

Adding data

Specifying the data type is not compulsory, but it can be useful.

In [3]:
```
dtype = tb.Float64Atom()
data = np.random.rand(10, 5)

with tb.open_file(fpath, mode='a', filters=filters) as h5_file:
    h5_file.create_carray(obj=data, atom=dtype, where='/', name='table_1')
    print(h5_file)
```

```
./test.h5 (File) ''
Last modif.: 'Wed Sep  9 18:49:50 2020'
Object Tree:
/ (RootGroup) ''
/table_1 (CArray(10, 5), shuffle, blosc:lz4hc(9)) ''
```

We have just saved the table `data` of shape `(10, 5)` into a dataset named `table_1` on the root group of the HDF5 file.

Dataset styles

Note that `pytables` allows to create groups and datasets in different ways:

- `create_group` : Create a new group.
- `create_array` : Create a new array.
- `create_carray` : Create a new chunked array.
- `create_earray` : Create a new enlargeable array.
- `create_table` : Create a new table. Haven't worked with them.

Reference:

- https://www.pytables.org/usersguide/libref/file_class.html?highlight=earray#tables.File.create_array (https://www.pytables.org/usersguide/libref/file_class.html?highlight=earray#tables.File.create_array)

Attributes

Adding attributes to groups or datasets is one of the key features HDF5 provides.

In [4]:
```
data = np.random.rand(15, 10)

with tb.open_file(fpath, mode='a', filters=filters) as h5_file:
    node = h5_file.create_carray(obj=data, atom=dtype, where='/', name='tabl
e_2')
    # Add meta: column names
    node._v_attrs['experiment_date'] = time.strftime('%d/%B/%Y %H:%M:%S', ti
me.localtime())
    print(h5_file)
    print()
```

```
./test.h5 (File) ''
Last modif.: 'Wed Sep  9 18:49:54 2020'
Object Tree:
/ (RootGroup) ''
/table_1 (CArray(10, 5), shuffle, blosc:lz4hc(9)) ''
/table_2 (CArray(15, 10), shuffle, blosc:lz4hc(9)) ''
```

We have just created a new chunked array and added some metadata to it. We can retrieve it as follows:

In [5]:
```
with tb.open_file(fpath, mode='r', filters=filters) as h5_file:
    print('Date of the experiment:', h5_file.get_node('/table_2')._v_attrs['
experiment_date'])
```

```
Date of the experiment: 09/September/2020 18:49:54
```

Generally, you can save any Python object as an attribute (lists, dictionaries...).

Reading from and HDF5 file

Reading the full table

There are different ways for reading data from an HDF5 file:

In [6]:
```
with tb.open_file(fpath, mode='r', filters=filters) as h5_file:
    print(h5_file.get_node('/table_1'))
```

```
/table_1 (CArray(10, 5), shuffle, blosc:lz4hc(9)) ''
```

Note that in the previous example, we havent read the data yet, but rather we have a reference to it. We can force it to read by adding `[:]` :

In [7]:
```
with tb.open_file(fpath, mode='r', filters=filters) as h5_file:
    data = h5_file.get_node('/table_1')[:]
    print(data)
```

```
[[0.31210029 0.97397513 0.50778398 0.86512333 0.86052477]
 [0.23769895 0.54661755 0.47077833 0.41527728 0.56254675]
 [0.74181591 0.77437575 0.97620274 0.05962677 0.03046987]
 [0.83718529 0.2730375  0.49913297 0.15654673 0.21424502]
 [0.48337156 0.51953183 0.7572829  0.86366161 0.7796142 ]
 [0.76353492 0.43932678 0.19088249 0.04277602 0.90556977]
 [0.30980525 0.90509908 0.89485527 0.14282516 0.72605461]
 [0.15176233 0.37530405 0.31438245 0.45979472 0.33488458]
 [0.7856728  0.04131435 0.09246083 0.50245168 0.45553898]
 [0.37581702 0.47238327 0.61691669 0.15876077 0.17415818]]
```

Slicing

Same as with `numpy`, we can slice the array. For example, the last row and starting from the 2nd column:

In [8]:
```
with tb.open_file(fpath, mode='r', filters=filters) as h5_file:
    print(h5_file.get_node('/table_1')[-1, 2:])
```

```
[0.61691669 0.15876077 0.17415818]
```

Note that this is very useful **if you don't want to read the full array into memory**.

Alternative way to read data

Instead of using `get_node`, you can also read data in similar fashion as with Python dictionaries:

In [9]:
```
with tb.open_file(fpath, mode='r', filters=filters) as h5_file:
    print(h5_file.root['/table_1'][-1, 2:])
```

```
[0.61691669 0.15876077 0.17415818]
```

Vitables

Vitables is a Python program that allows the visualization (and, to some extent, manipulation) of HDF5 files. It can be `pip` installed as usual. The previous test dataset can be opened and data can be displayed, including its attributes.

Reference: https://vitables.org/ (https://vitables.org/)