h5py - Eine Python Implementation von HDF5

Proseminar: Python im Hochleistungsrechnen

Melvin Höfges

Fachbereich Informatik

27.Mai.2019

Gliederung

Was ist das ? Wofür wird es genutzt ?

- Was ? Implementation des HDF5 Datenformats in Python
- Wofür ? Bearbeitung und Speicherung von riesigen Datenmengen in Numpy-arrays
- Wo ? Open source auf GitHub [git]
- Homepage: https://www.h5py.org/ [hom]
- Wer ? Andrew Collette und 6 weitere

Verwendung

- Vorteile:
  - 'Metadaten' an Datensätze anheften
  - Unterstützung sehr großen Datensätzen
  - Verfügbar auf verschiedensten Systemen
  - Hohe Performanz
  - Einfache Weitergabe von Daten
  - Kompatibel mit vielen etablierten Programmen
- Verwendet von:
  - Flugzeugindustrie - Design und Test Standard
  - Automobilindustrie - Design und Test Datensammlung
  - Finanzwelt - Auswertung von Börsendaten
  - Und viele weitere

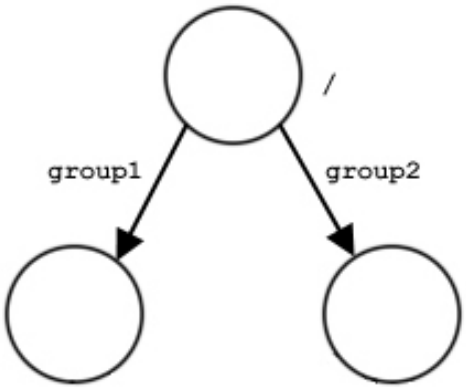
Quelle: www.hdfgroup.org [HDF]

Installation

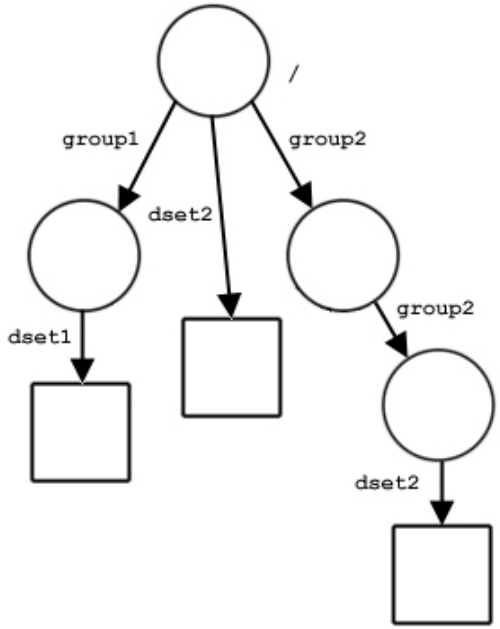
- Voraussetzung :
  - Python 2 / 3
  - NumPy
- Installation :
  - via Conda oder PIP

```
>>> pip install h5py
```

HDF5 Datenstruktur



(a) Rootverzeichnis mit 2 groups



(b) Verzeichnis mit insgesamt 3 groups und 3 datasets

Quelle: davis.lbl.gov/Manuals/HDF-1.8.7/

Grundlegende Objekte

- HDF5 Dateien

```
>>> f = h5py.File('Name.hdf5', 'w')
/*   w create or overwrite,
     r read,
*/   a read/write if exists or create (default)
```

- groups
  - Ordner ähnliche Strukturen mit Attributen

```
>>> grp = f.create_group("Name")
```

- datasets
  - NumPy arrays mit Attributen

```
>>> dset = f.create_dataset("Name", (Werte),
    ↪ dtype='Typ'))
```

Strings

- ASCII und UTF-8
- Objektnamen dürfen auch Unicode Zeichen enthalten
- Für Binärdaten soll der Typ 'void' benutzt werden !

Zitat aus der Dokumentation:

'If you remember nothing else, remember this:

All strings in HDF5 hold encoded text.

You can't store arbitrary binary data in HDF5 strings. Not only will this break, it will break in odd, hard-to-discover ways that will leave you confused and cursing.'

Attribute

- 'Metadaten'
- Objektname.Befehl()

```
>>> keys() - Namen aller Attribute des Objekts
```

```
>>> values() - Werte aller Attribute des Objekts
```

```
>>> items() - Tupel mit Namen und Wert aller
    ↪ Attribute
```

```
>>> create(Name, Werte, Form, Typ)
```

```
>>> modify(Name, Werte)
```

Befehl Demo

Auf zur Konsole

Datenmenge als Problem

- Zu viele Daten für direktes Speichern
- Datenverlust bei abstürzen
- Niedrige Performanz bei steigenden Dateigrößen

Parallel HDF5

- Bearbeiten von Dateien über mehrere Prozesse / Threads
- MPI - Message Passing Interface

```
1 //demo.py
2 >>> from mpi4py import MPI
3 >>> print "Hello World (from process %d)" %
    ↪ MPI.COMM_WORLD.Get_rank()
```

```
1 $ mpiexec -n 3 python demo.py
2 Hello World (from process 1)
3 Hello World (from process 2)
4 Hello World (from process 0)
```

Quelle: h5py Dokumentation [MPI]

SWMR - Single Writer Multiple Reader

- 1 Writer Schreibt / Liest eine Datei
- mehrere Reader werden über Änderungen Informiert

- Failsafe bei Crash des Writerprozesses
- Parallele Verteilung durch Virtual Datasets

SWMR - Writer

```
1  >>> f = h5py.File("swmr.h5", 'w', libver='latest')
2  >>> arr = np.array([1,2,3,4])
3  >>> dset = f.create_dataset("data", chunks=(2,),
       ↪ maxshape=(None,), data=arr)
4  >>> f.swmr_mode = True
5  # Now it is safe for the reader to open the swmr.h5
       ↪ file
6  >>> for i in range(5):
7          new_shape = ((i+1) * len(arr), )
8          dset.resize( new_shape )
9          dset[i*len(arr):] = arr
10         dset.flush()
11 # Notify the reader process that new data has been
       ↪ written
```

Quelle: h5py Dokumentation [SWM]

SWMR - Reader

```
1 >>> f = h5py.File("swmr.h5", 'r', libver='latest',
      ↪ swmr=True)
2 >>> dset = f["data"]
3 >>> while True:
4         dset.id.refresh()
5         shape = dset.shape
6         print( shape )
```

Quelle: h5py Dokumentation [SWM]

VDS - Virtual Datasets

- Nicht Langfristig gespeichert
- Mittel zur Aufteilung von großen Datasets
- Mittel zur Zusammenführung verschiedener Datasets

VDS - Beispiel Teil 1

```
1  # Create source files (1.h5 to 4.h5)
2  for n in range(1, 5):
3      with h5py.File('{}.h5'.format(n), 'w') as f:
4          d = f.create_dataset('data', (100,), 'i4')
5          d[:] = np.arange(100) + n
6
7  # Assemble virtual dataset
8  layout = h5py.VirtualLayout(shape=(4, 100),
       ↪ dtype='i4')
9
10 for n in range(1, 5):
11     filename = "{}.h5".format(n)
12     vsource = h5py.VirtualSource(filename, 'data',
           ↪ shape=(100,))
13     layout[n - 1] = vsource
```

VDS - Beispiel Teil 2

```
1 // Add virtual dataset to output file
2 with h5py.File("VDS.h5", 'w', libver='latest') as f:
3     f.create_virtual_dataset('data', layout,
      ↪ fillvalue=-5)
4     print("Virtual dataset:")
5 print(f['data'][:, :10])
```

Quelle: GitHub(h5py/h5py/blob/master/examples/

Zusammenfassung

- HDF5
  - groups und datasets zum Organisieren und Speichern
  - Attribute zum Anfügen von 'Metadaten'
- Geschwindigkeit
  - Erreicht in Rechenclustern bzw. Systemen mit viel Möglichkeit zum Parallel arbeiten
  - Effizienz kommt durch Aufteilung

Literatur

SWMR-Beispielcode: github.com/h5py/...
VDS-Besispielcode: github.com/h5py/...

[git] https://github.com/h5py/h5py.

[HDF] https://www.hdfgroup.org/solutions/hdf5.

[hom] https://www.h5py.org/.

[MPI] http://docs.h5py.org/en/stable/mpi.html.

[SWM] http://docs.h5py.org/en/stable/swmr.html.