Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Ausarbeitung

h5py - Eine Python Implementation von HDF5

vorgelegt von

Melvin Höfges

Fachbereich Informatik

Studiengang: Informatik
Matrikelnummer: 7063214

Betreuer: Kira Duwe

Köthel 2019-08-31

Inhaltsverzeichnis

1 Einführung in HDF5 und h5py

1.1 Allgemeines

h5py ist eine Implementation des HDF5 Datenformats in Python. HDF5 ist ein selbstbeschreibendes Datenformat zur Speicherung und Bearbeitung riesiger Datenmengen, als wären diese typische NumPy-arrays. An jeden Datensatz können Metadaten angehängt werden um Daten zu erklären, in Kontext zu setzen oder beispielsweise ihren Aufbau und Erhebung zu klären. Dies erhöht die Lesbarkeit von Datensätzen für Nutzer und vereinfacht eine automatisierte Datenverarbeitung, da über die Metadaten Information an Software gegeben werden kann, um deren Speicherung oder Verarbeitung zu steuern. Selbstbeschreibende Datenformate sind in vielen Bereichen der Datenverarbeitung zum Standard geworden. Mit Formaten wie JSON, XML oder Apache Parquet ist das Prinzip von Selbstbeschreibenden Datenformaten bereits weit verbreitet und wird besonders im $Internet of Things$ verwendet um Daten zwischen verschiedensten Geräten und Dienstleistern zu teilen.

HDF5 wird hauptsächlich in wissenschaftlichen Instituten zur Datenanalyse verwendet, vereinzelt kommt es aber auch in verschiedenen Teilen der Industrie oder an Börsen vor. Allgemein lässt sich sagen, dass HDF5 überall dort Anwendung findet, wo es große Datenmengen gibt, die es möglichst einfach und schnell zu verarbeiten und etwa zu kategorisieren gilt.

Gegründet von Andrew Colette und 6 weiteren, besitzt das Projekt 2019 mehr als 100 Mitwirkende auf Github. Das erste Release von h5py auf PyPI war am 3. Dezember 2008. Es handelt sich um eine Open-Source-Entwicklung auf Github, die noch heute mehrere Updates im Monat erhält. Die Entwicklung ist allerdings größtenteils abgeschlossen.

1.2 Installation

Zur Installation von h5py wird Python 2.x oder 3.x sowie NumPy vorausgesetzt, ein Packagemanager wie Conda oder PIP wird dringend empfohlen.

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

1.3 Datenstruktur

In HDF5 Dateien gibt es zwei grundsätzliche Strukturen *Gruppen*(*engl.* : *groups*) und *Datensätze*(*engl.* : *datasets*). Gruppen sind als Ordner zu verstehen und können weitere Gruppen oder Datensätze beinhalten. Datensätze sind NumPy-array ähnliche Strukturen, die Daten speichern können. Sowohl Gruppen als auch Datensätze besitzen *Attribute*, die dazu genutzt werden können, um Metadaten zu speichern.

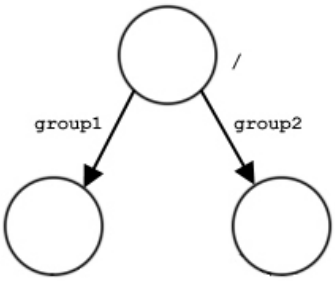

Abbildung 1.1: Rootverzeichnis mit 2 Gruppen.
$http://davis.lbl.gov/Manuals/HDF5-1.8.7/UG/03\_DataModel.html$

In Abbildung 1.1 sieht man das Rootverzeichnis oben und 2 Gruppen darunter.

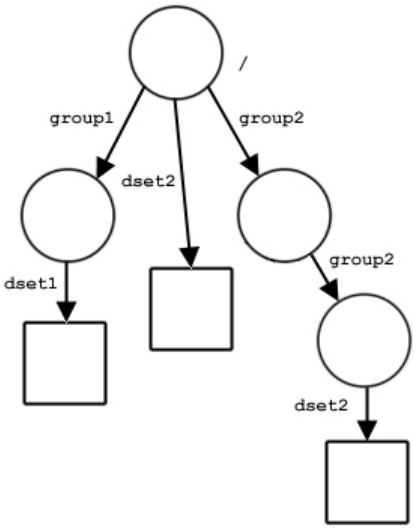

Abbildung 1.2: Verzeichnis mit insgesamt 3 Gruppen und 3 Datensätzen
$http://davis.lbl.gov/Manuals/HDF5-1.8.7/UG/03\_DataModel.html$

In Abbildung 1.2 sieht man das Rootverzeichnis oben. Dieses Verzeichnis beinhaltet 2 Gruppen und den Datensatz *dset*2. In *group*1 befindet sich der Datensatz *dset*1. *group*2 beinhaltet eine weitere Gruppe mit gleichen Namen, in welchem sich ein Datensatz mit dem Namen *dset*2 befindet.
Der Pfad zum rechten *dset*2 ist also : '/group2/group2/dset2'

1.4 Dateien erstellen und bearbeiten

Das Erstellen und Öffnen einer Datei funktioniert bei h5py mit dem `.File` Befehl, dieser bekommt 2 Argumente.

Das erste Argument ist der Dateiname inklusive Pfad.
Das zweite Argument der Modus. Dieser wird mit einem der Buchstaben 'w, r oder a' ausgewählt.

Mit 'w' wird eine Datei erstellt. Falls eine Datei mit selben Namen bereits vorhanden ist, wird diese überschrieben.
Mit 'r' wird eine Datei nur zum Lesen geöffnet.
Mit 'a' wird eine Datei geöffnet und kann bearbeitet werden. Falls keine Datei mit angegebenen Namen existiert, wird eine neue erstellt.

```
1 >>> f = h5py.File('Name.hdf5', 'w')
2 /*
3 w create or overwrite, r read,
4 a read/write if exists or create (default)
5 */
```

Sobald eine Datei geöffnet oder erstellt wurde, können *Gruppen* und *Datensätze* in dieser erstellt werden. *Gruppen* können direkt oder indirekt erstellt werden.

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

Listing 1.1: Direkte Erstellung einer Gruppe

Hier wurde die Gruppe *Name* direkt im Rootverzeichnis der Datei erstellt.

```
1 >>> grp = f.create_group("/group1/group2/Name")
```

Listing 1.2: Indirekte Erstellung von Gruppen

Hier wurden die Gruppen *group*1 und *group*2 indirekt bei der Erstellung der Gruppe *Name* mit erstellt.

Datensätze können auf gleiche weise direkt im aktuellen Verzeichnis erstellt werden. Durch einen Pfad im *Name* Argument können Gruppen indirekt mit erstellt, beziehungsweise zu geöffnet, werden.

```
1 >>> dset = f.create_dataset("Name", (Form), dtype='Typ')
```

Der Befehl besitzt neben dem wie bei Gruppen funktionierenden Argument *Name* noch die 2 Argumente *Form* und *dtype*.
Unter `Form` ist hier bei die Dimensionierung des Datensatzes zu verstehen und wird wie bei einem Array in NumPy angegeben.
`dtype` gibt an vom welchen Typ die gespeicherten Werte sein sollen, wie zum Beispiel 'i8' was einem signed 8byte Integer entsprechen würde.

1.5 Attribute

Mit Hilfe von Attributen lassen sich in HDF5 Dateien Metadaten an Gruppen und Datensätzen anbringen. Jede Gruppe und jeder Datensatz besitzt von Erstellung an ein *Proxyobject* mit dem Namen `attrs`. Dieses Objekt ist dafür zuständig die Attribute zu sammeln und an die Gruppen oder den Datensatz zu binden. Die wichtigsten Befehle an dem `attrs` Objekt sind :

```
1 dset.attrs.keys()
2 dset.attrs.items()
3 dset.attrs.get('name')
4 dset.attrs.create('name', data)
```

Listing 1.3: Befehle am *attrs* Objekt

`keys()` gibt die Namen aller bestehenden Attribute an dem dataset 'dset' zurück.

`items()` gibt für jedes bestehende Attribut ein Tupel mit Namen und Wert zurück

`get('Name')` gibt den Wert des Attributs Name wieder falls vorhanden.

`create('Name', data)` Erstellt ein neues Attribut Name mit data als Wert, standardmäßig ist data ein NumPy String und lässt sich über das Argument dtype=' ' verändern.

2 SWMR - Single Writer Multiple Reader

2.1 Allgemeines

Der SWMR Modus in h5py erlaubt es, eine HDF5 Datei zu erstellen und zu bearbeiten, während beliebig viele Reader Prozesse die Datei lesen können.
Dies bedeutet, dass sich die Datei immer in einem validen Zustand befindet und im Fall eines Absturzes des Writer Prozesses nicht die Gefahr besteht, dass es zu Datenverlust kommt.
Sobald der SWMR Modus aktiv ist, werden alle Reader über Änderungen in der Datei informiert, sodass diese ihre Version der Datei aktualisieren können. Dies kann abhängig von den beteiligten Systemen unterschiedlich lange dauern, behindert nach der Benachrichtigung aber nicht mehr den Writer Prozess, sodass dieser schnell wieder seinen Aufgaben nachkommen kann.

2.2 Nutzung

Sobald eine HDF5 Datei geöffnet ist, kann mit dem Befehl in Listing 2.1 der SWMR Modus aktiviert werden.

```
1 f.swmr_mode = True
```

Listing 2.1: 'SWMR Modus aktivieren'

Sobald das passiert ist, kann ein weiterer Prozess sich die Datei ansehen mit :

```
1 f = h5py.File("Name.h5", 'r', libver='latest', swmr=True)
```

Listing 2.2: 'SWMR Reader'

Sei in der Datei Name.h5 ein dataset mit dem Namen 'data' im Rootverzeichnis so könnte der Reader Prozess dieses nun öffnen und live aktualisieren.

```
1 dset = f["data"]
2 while True:
3     dset.id.refresh()
4     # weitere Anweisungen
```

Listing 2.3: 'SWMR aktualisieren'

3 VDS - Virtual Datasets

3.1 Allgemeines

Virtual Datasets ermöglichen es, mehrere Datensätze zu einem *virtualdataset* zusammen zu führen ohne die Originalen Datensätze zu verändern.
Zum Erstellen eines *virtualdatasets* sind die Datensätze, die zusammen geführt werden sollen, zu Beginn nicht nötig. Somit kann ein *virtualdataset* vorbereitet werden, bevor die tatsächlichen Datensätze vorliegen.

Sobald ein *virtualdataset* erstellt wurde, kann es wie ein normaler Datensatz gelesen werden. Zum Bearbeiten muss das *virtualdataset* als ein neues eigenständiger Datensatz gespeichert werden.

3.2 Anwendung

Zuerst werden 4 Dateien erstellt, die jeweils ein eindimensionalen Datensatz mit dem Namen *data* beinhalten. In jedem Datensatz stehen jeweils 100 Zahlen beginnend bei der Dateinummer.

```
1 for n in range(1, 5):
2     with h5py.File('{}.h5'.format(n), 'w') as f:
3         d = f.create_dataset('data', (100,), 'i4')
4         d[:] = np.arange(100) + n
```

Listing 3.1: 'VDS create source files'

```
1 Dataset:1
2 [  1   2   3   4   5   6   7   8   9  10  ...  91  92  93
    ↪ 94  95  96  97  98  99 100]
```

Listing 3.2: '1.h5 output'

Jetzt wird ein zweidimensionales *virtualdatasetlayout* erstellt mit den Dimensionsgrößen 4 und 100.

```
1 layout = h5py.VirtualLayout(shape=(4, 100), dtype='i4')
```

Listing 3.3: 'VDS virtual dataset layout'

Sobald das Layout erstellt wurde, können die einzelnen Dateien in das Layout geladen werden. Sie nehmen dabei jeweils eine der 4 Spalten ein.

```
1 for n in range(1, 5):
2     filename = "{}.h5".format(n)
3     vsource = h5py.VirtualSource(filename, 'data',
        ↪ shape=(100,))
4     layout[n - 1] = vsource
```

Listing 3.4: 'VDS virtual dataset merge'

```
1 Virtual dataset:
2 [[  1   2   3   4   5   6   7   8   9  ...   91  92  93  94
    ↪ 95  96  97  98  99 100]
3  [  2   3   4   5   6   7   8   9  10  ...   92  93  94  95
    ↪ 96  97  98  99 100 101]
4  [  3   4   5   6   7   8   9  10  11  ...   93  94  95  96
    ↪ 97  98  99 100 101 102]
5  [  4   5   6   7   8   9  10  11  12  ...   94  95  96  97
    ↪ 98  99 100 101 102 103]]
```

Listing 3.5: 'VDS merge output'

Zum Abschluss kann jetzt das *virtualdataset* als eigene Datei gespeichert werden, um eine weitere Bearbeitung möglich zu machen.

```
1 with h5py.File("VDS.h5", 'w', libver='latest') as f:
2     f.create_virtual_dataset('data', layout, fillvalue=-5)
```

Listing 3.6: 'VDS complete File'

VDS Besispielcode: *https* : *//github.com/h5py/h5py/blob/master/examples/vds_simple.py*

4 Zusammenfassung

HDF5 ist ein selbstbeschreibendes Dateiformat, das bei dem Organisieren und Verarbeiten großer Datenmengen hilft und mit vielen, in der Forschung zum Standard erklärten, Programmen kompatibel ist. Durch die Fähigkeit sehr große Datensätze in eine einfach überschaubare Datei zu bündeln, wird das Teilen von Daten stark vereinfacht.
Die Implementation h5py bringt dabei viele Tools, mit denen zum Beispiel Arbeitsaufteilung in Rechenclustern oder angenehmeres Organisieren und Zusammenführen von Daten ermöglicht und vereinfacht werden.
Hohe Performanz wird mit h5py, in erster Linie, durch Aufteilung und Parallelisierung erreicht. Für die Nutzung an einem Rechner liegt also der Vorteil eher in der Organisation und Teilbarkeit.

5 Quellen

h5py homepage (besucht am 14.5.2019):
https : *//www.h5py.org/*

h5py Dokumentation von Andrew Colette und Mitwirkenden (besucht am 14.5.2019):
http : *//docs.h5py.org/en/stable/*

HDF5 Nutzeranleitung(engl.) von der HDF Group und der University of Illinois (besucht am 26.5.2019):
http : *//davis.lbl.gov/Manuals/HDF*5 − 1.8.7/*UG/UG_frame*08*TheFile.html*

Datenstrukturbilder der HDF Group und der University of Illinois (besucht am 26.5.2019):
http : *//davis.lbl.gov/Manuals/HDF*5 − 1.8.7/*UG*/03_*DataModel.html*

SWMR aus der Dokumentation von Andrew Colette und Mitwirkenden (besucht am 16.5.2019):
http : *//docs.h5py.org/en/stable/swmr.html*

VDS aus der Dokumentation von Andrew Colette und Mitwirkenden (besucht am 16.5.2019):
http : *//docs.h5py.org/en/stable/vds.html*

VDS Besispielcode (besucht am 16.5.2019):
https : *//github.com/h5py/h5py/blob/master/examples/vds_simple.py*

Self-describing data formats and their growing role in analytics... von Neeraja Rentachintala (besucht am 20.5.2019): *https* : *//mapr.com/blog/evolving* − *parquet* − *self* − *describing* − *data* − *format* − *new* − *paradigms* − *consumerization* − *hadoop* − *data/*