HDF5 - standard plików BIG DATA

1. Co to jest ?

HDF5 (Hieracchial Data Format v5) - jest to standard opisujący strukturę pliku, do którego w hierarchiczny i ustrukturyzowany sposób możemy zapisywać tablicę danych (system plików w jednym pliku). Dostęp do danych zgromadzonych w datasecie możemy dostać się poprzez interfejs identyczny z **numpy**. Jest to istotne, bo operacje odbywają się na dysku, do pamięci wczytane są dane, które są potrzebne w danym momencie. Kto wczyta do pamięci 80GB plik ?

---

2. W jakim celu został utworzony?

Worklow podczas pracy związanej z analizą danych: odczyt danych -> wczytanie do pamięci -> algorytmy ->dobieramy parametry -> zbieramy dane.
Przetwarzanie ogromnych plików z danymi wymagania dużej wolnej przestrzeni w RAM'ie, w przypadku bardzo dużych plików danych może nam tej pamięci zabraknąć, co wtedy? Aby uniknąć tego problemu należy dane wczytywać i przetwarzać dane partiami. Takie podejście wymaga dodatkowego kodu, a HDF5 umożliwia łatwe i schludne napisanie kodu.

---

3. Format pliku

W obrębie pliku możemy utworzyć 3 rodzaje obiektów:

- Grupy - tak jak foldery w systemie plików
    - Datasety - tak jak pliki w systemie plików, ale mają postać macierzy o ustalonych wymiarach
        - Atrybuty - metadane, które umożliwiają opisanie grupy lub datasetu(dodatkowa informacja np. autor, data itd.)

---

4. W czym pomaga?

- Zbieranie i przechowywanie danych z czujników/sensorów w formie jednego pliku, a w nim możemy po katalogować dane, dodatkowo opisać poszczególne partie.
- Przetwarzania małych plików. Zamiast pracować z milionami plików, możemy je zapisać w jednym pliku. Dzięki temu łatwiej możemy taki plik łatwiej udostępnić, operacje kopiowania z dysku na dysk lub przez sieć wykonują się zdecydowanie szybciej.
- Zgodność interfejsu z **numpy*

---

5. Przechowywanie w blokach

Dane możemy również przetrzymać w blokach/częściach. Dataset utworzony przy użyciu **HDF5's chunked layout** jest podzielony na regularnego rozmiaru części i jest przechowywany na dysku w różnych miejscach dostęp odbywa się za pomocą indeksowania B-tree. Umożliwi to zmianę rozmiaru datasetów, ponieważ dane są przechowywane w stałych chunkach oraz wykorzystanie skompresowanego filtrowania. Rekomendowany rozmiar chunka 10KB - 1MB.

---

6. Przykłady

Z użyciem pythonwej biblioteki **h5py**.

1. Otwieranie i tworzenie plików

```
import h5py

# Poprawne rodzaje: r(ro, plik musi istnieć), r+(rw), w(tworzy plik), w-
(x)(tworzy plik, jeśli istnieje błąd), a(rw, tworzy jeśli nie ma -
domyślnie)
hdf_file = h5py.File('myfile.hdf5','r')
```

2. Tworzenie grup:

```
import h5py

group = hdf_file.create_group("groupa")
sub_group = group.crate_group("podgrupa")
# Grupy korzystają ze słownikowej konwencji pythona
group["podgrupa"]
# Usuwanie
del group["podgrupa"]
```

3. Lekkie dowiązania do grup:

```
import h5py

yfile = h5py.File('foo.hdf5','w')
group = myfile.create_group("somegroup")
myfile["alias"] = h5py.SoftLink('/somegroup')

# Usunięcie spowoduje błędy
del myfile['somegroup']
print myfile['alias']
# KeyError: 'Component not found (Symbol table: Object not found)'
```

4. Tworzenie datasetów:

```
import h5py
import numpy as np

# h5py wspiera większość typów numpy
dataset = hdf_file.create_dataset(name="name", shape=(100,), dtype="f")
dataset_copy = hdf_file["name"]

# Można użyć np.array() zamiast shape i dtype
array = np.arrange(100)
dataset = hdf_file.create_dataset(name="name", data=array)
```

5. Pisanie w blokach/częściach(chunks)

```
import h5py

# Dane będą trzyamne w blokach 100 x 100
dataset = hdf_file.create_dataset(name="chunked", (1000, 1000), chunks=
(100,100))
# Automatyczny dobór rozmiaru chunka
dataset = hdf_file.create_dataset("autochunk", (1000, 1000), chunks=True)
```

6. Dataset ze zmiennym rozmiarem

```
import h5py

# Z ograniczeniem maksymalnego rozmiaru
dataset = hdf_file.create_dataset("resizable", (10,10), maxshape=(500,
20))
# Potencjalnie nieograniczone(2**64)
dataset = hdf_file.create_dataset("resizable", (10,10), maxshape=(None,
10))
# Zmiana rozmiaru nie jest zaimplementowana tak jak w numpy, jeśli któraś
os się
#  skurczy, dane w brakującej części są odrzucane, nie nastpuję
przeorganizowanie danych.
dataset.resize(new_size, axis)
```

7. Filter pipline

```
import h5py

# Dane w blokach mogą być przekształcane przy pomcy filter pipiline. Dane
są
#  skompresowane na dysku i automatycznie dekompresowane gdy następuje
odczyt.
# Jeśli dane są utworzone z filtrem kompresji, Dalej można z nich
korzystać jak ze zwykłego datasetu
datset = hdf_file.create_dataset("zipped", (100, 100), compression="gzip")
# Można dodać opcje
datset = hdf_file.create_dataset("zipped", (100, 100), compression="gzip",
compression_opts=9)
# Filtry straty kompresji: gzip (opts to poziom kompresji 0 - 9), lzf(nie
ma opts), szip
# (no opts). Można używac customowych filtrów.
# Scale-Offset filter - skaluje do określonego typu danych
scaleoffset="i8"
# Shuffle filter - gzip i lzf lepiej działają na podobnych danych, włącz
ten filtr,
# aby zoptymalziować shuffe=True
# Fletcher32 filter - sprawdzanie sum kontrolnych fletcher32=True
```

8. Czytanie i zapisywanie danych tak samo jak w numpy. Broadcasting jest dostępny.
9. Fancy indexing również jest dostępne.
10. Arybuty - można dodawać do grupy i zbiorów danych

```
import h5py

dataset = hdf_file.create_dataset("resizable", (10,10), dtype="f"))
group = hdf_file.create_group("groupa")

dataset.attr["atyrbut1"] = "wartość1"
group.attr["atrybut2"] = "wartość2"
```

Właściwości atrybutów:

- Mogą być tworzone w dowolnej tablicy skalarnej lub NumPy
- Każdy atrybut powinien być mały mniejszy niż 64K
- Nie ma częściowego I/O , cały atrybut musi zostać odczytany.

Domyślnie po atrybutach iteruje się w alfabetyczny porządku,, jeśli grupa została stworzona z `track_order=True`, kolejność dodania jest zapamiętywana. Można ustalić wartość dla wszystkich poprzez:
`h5.get_config().track_order`

7. HDF5 w Tensorflow

W tensorflow jest dostępna klasa `tf.keras.utils.HDF5Matrix`, która wspiera zbiory danych HDF5, aczkolwiek jest przestarzała i trenowanie modeli z taką macierzą może okazać się niezoptymalizowane pod kątem wydajności i może nie działać z każdą strategią dystrybucji. Zaleca się użycie tensorflow.io żeby załadować dane z pliku .hdf5 do tf.data i przekazać do Kerasa. https://www.tensorflow.org/io/api_docs/python/tfio/v0/IODataset#from_hdf5

```
tfio.IODataset.from_hdf5("nazwa_pliku", "dataset_name", spec, name)
# spec - tf.TensorSpec lub dtype zbioru danych. W trybie grafu, spec jes
wymagany.
# W trybie eager jest sondowany automatycznie.
# name - prefix nazwy dla IOTensor (opcjonalnie)
```

Bibliografia

- https://ksopyla.com/data-science/big-data-na-dysku-czyli-jak-przetwarzac-pliki-hdf5-w-python/