

Dask Array

Материалы:

- Макрушин С.В. Лекция 11: Dask
- <https://docs.dask.org/en/latest/array.html> (<https://docs.dask.org/en/latest/array.html>)
- JESSE C. DANIEL. Data Science with Python and Dask.

Задачи для совместного разбора

1. Создайте массив размерностью 1000 на 300000, заполненный числами из стандартного нормального распределения. Исследуйте основные характеристики полученного массива.
2. Посчитайте сумму квадратов элементов массива, созданного в задаче 1. Создайте массив `np.array` такого же размера и сравните скорость решения задачи с использованием `da.array` и `np.array`
3. Визуализируйте граф вычислений для задачи 12.

Лабораторная работа 11

```
In [1]: import os
import dask.array as da
import h5py
import numpy as np
```

1. Считайте датасет `recipe` из файла `minutes_n_ingredients_full.hdf5` в виде `dask.array`. Укажите аргумент `chunks=(100_000, 3)` при создании массива. Выведите на экран основную информацию о массиве.

```
In [2]: data_f = h5py.File(r'C:\Users\sunya\Desktop\6семестр\интернетвещей\dask\minutes_n_ingredients_full.hdf5')
list(data_f.keys())
```

```
Out[2]: ['recipe']
```

```
In [3]: data_f['recipe'].shape
```

```
Out[3]: (2231637, 3)
```


```
In [4]: da_ar = da.from_array(data_f['recipe'], chunks=(100_000, 3))

# r = da_ar - da_ar.mean(axis=0)
```

```
In [5]: da_ar
```

```
Out[5]:
```

	Array	Chunk
Bytes	51.08 MiB	2.29 MiB
Shape	(2231637, 3)	(100000, 3)
Dask graph	23 chunks in 2 graph layers	
Data type	int64 numpy.ndarray	3



2. Вычислите среднее значение по каждому столбцу, кроме первого.

```
In [6]: x_np = np.array(data_f['recipe'])
x_np.mean(axis=0)
```

```
Out[6]: array([1.12684089e+06, 1.00420805e+03, 5.41980080e+00])
```

```
In [7]: da_ar[:, 1:].mean(axis=0).compute()
```

```
Out[7]: array([1004.20805176,    5.4198008 ])
```

3. Исследуйте, как влияет значение аргумента `chunks` при создании `dask.array` на скорость выполнения операции поиска среднего.

```
In [8]: da_ar = da.from_array(data_f['recipe'], chunks=(100_000, 3))
```

```
In [9]: %%time
da_ar.mean(axis=0).compute()
```

```
CPU times: total: 93.8 ms
Wall time: 39.4 ms
```

```
Out[9]: array([1.12684089e+06, 1.00420805e+03, 5.41980080e+00])
```

```
In [10]: da_ar = da.from_array(data_f['recipe'], chunks=(300_000, 3))
```

```
In [11]: %%time
da_ar.mean(axis=0).compute()
```

CPU times: total: 93.8 ms

Wall time: 35.3 ms

```
Out[11]: array([1.12684089e+06, 1.00420805e+03, 5.41980080e+00])
```

```
In [12]: da_ar = da.from_array(data_f['recipe'], chunks=(10_000, 3))
```

```
In [13]: %%time
da_ar.mean(axis=0).compute()
```

CPU times: total: 78.1 ms

Wall time: 73.1 ms

```
Out[13]: array([1.12684089e+06, 1.00420805e+03, 5.41980080e+00])
```

чем больше размер chunks, тем быстрее считается среднее

4. Выберите рецепты, время выполнения которых меньше медианного значения

```
In [14]: import dask.dataframe as dd
import pandas as pd

df1 = pd.DataFrame(da_ar)
df1
```

Out[14]:

	0	1	2
0	683970	33	9
1	1089012	23	5
2	1428572	0	5
3	1400250	24	1
4	387709	47	10
...
2231632	1029131	19	4
2231633	1700703	1	1
2231634	1910650	60	2
2231635	713836	0	9
2231636	660699	64	8

2231637 rows × 3 columns

```
In [15]: df = dd.from_pandas(df1, chunksize=10000)
df
```

Out[15]: **Dask DataFrame Structure:**

	0	1	2
npartitions=224			
0	int64	int64	int64
10000
...
2230000
2231636

Dask Name: from_pandas, 1 graph layer

```
In [16]: median_time = df1[1].median(axis=0)
median_time
```

Out[16]: 32.0

```
In [17]: df[df[1] < median_time]
```

Out[17]: **Dask DataFrame Structure:**

	0	1	2
npartitions=224			
	0	int64	int64
	int64	int64	int64
10000
...
2230000
2231636

Dask Name: getitem, 4 graph layers

```
In [18]: df1[df1[1] < median_time]
```

Out[18]:

	0	1	2
1	1089012	23	5
2	1428572	0	5
3	1400250	24	1
5	1798295	29	5
8	818815	21	5
...
2231628	1560061	14	7
2231630	2177253	29	7
2231632	1029131	19	4
2231633	1700703	1	1
2231635	713836	0	9

1084304 rows × 3 columns

5. Посчитайте количество каждого из возможных значений кол-ва ингредиентов

```
In [19]: df.groupby(2).sum()
```

Out[19]: **Dask DataFrame Structure:**

```
          0      1
npartitions=1
-----
      int64  int64
      ...    ...
```

Dask Name: dataframe-groupby-sum-agg, 3 graph layers

```
In [20]: df[2].unique()
```

Out[20]: **Dask Series Structure:**

```
npartitions=1
      int64
      ...
```

Name: 2, dtype: int64

Dask Name: unique-agg, 4 graph layers

```
In [21]: df1[2].value_counts()
```

```
Out[21]: 7      247181
          9      246816
          8      246747
          6      244360
          5      240720
          4      234948
          3      229388
          2      224158
          1      222071
         10      22430
         11      19094
         12      15165
         13      11640
         14       8284
         15       6014
         16       4145
         17       2793
         18       1913
         19       1279
         20        852
         21        529
         22        346
         23        244
         24        178
         25        107
         26         68
         27         55
         28         33
         29         22
         30         20
         31         13
         32          5
         35          4
         33          4
         34          3
         37          2
         40          2
         43          1
         39          1
         38          1
         36          1
Name: 2, dtype: int64
```

6. Найдите максимальную продолжительность рецепта. Ограничьте максимальную продолжительность рецептов сверху значением, равному 75% квантилю.

```
In [22]: df[1].max().compute()
```

```
Out[22]: 2147483647
```

```
In [23]: q = df[1].quantile(q=0.75).compute()
```

```
In [24]: df1[df1[1] < q]
```

Out[24]:

	0	1	2
0	683970	33	9
1	1089012	23	5
2	1428572	0	5
3	1400250	24	1
4	387709	47	10
...
2231630	2177253	29	7
2231631	1994055	47	9
2231632	1029131	19	4
2231633	1700703	1	1
2231635	713836	0	9

1679864 rows × 3 columns

7. Создайте массив `dask.array` из 2 чисел, содержащих ваши предпочтения относительно времени выполнения рецепта и кол-ва ингредиентов. Найдите наиболее похожий (в смысле L_1) рецепт из имеющихся в датасете.

```
In [25]: ar = da.array([12, 3])
ar
```

Out[25]:


	Array	Chunk
Bytes	8 B	8 B
Shape	(2,)	(2,)
Dask graph	1 chunks in 1 graph layer	
Data type	int32 numpy.ndarray	



In [26]: `da_ar`

Out[26]:

	Array	Chunk	
Bytes	51.08 MiB	234.38 kiB	
Shape	(2231637, 3)	(10000, 3)	
Dask graph	224 chunks in 2 graph layers		
Data type	int64	numpy.ndarray	3



In [45]: `da_ar_new = da_ar[:10000]`

In [49]: `da_ar_new.shape`

Out[49]: (10000, 3)

In [50]: `%%time`
`ar_tile = da.tile(ar, (da_ar_new.shape[0], 1))`

CPU times: total: 2.25 s
 Wall time: 2.25 s

In [51]: `diff = da.abs(da_ar_new[:, 1:] - ar_tile).sum(axis=1)`

In [54]: `data_f['recipe'][diff.argmin().compute()]`

Out[54]: array([2130931, 12, 3], dtype=int64)

8. Работая с исходным файлом в формате hdf5 , реализуйте алгоритм подсчета среднего значения в блочной форме и вычислите с его помощью среднее значение второго столбца в массиве.

Блочный алгоритм вычислений состоит из двух частей:

1. Загрузка фрагмента за фрагментом данных по `blocksize` элементов и проведение вычислений на этом фрагменте
2. Агрегация результатов вычислений на различных фрагментах для получения результата на уровне всего набора данных

Важно: при работе с `h5py` в память загружаются не все элементы, а только те, которые запрашиваются в данный момент

```
In [59]: blocksize = 10000

with h5py.File('minutes_n_ingredients_full.hdf5', 'r') as f:

    ds = f['recipe']
    s=0

    for i in range(0, ds.shape[0], blocksize):
        block = ds[i:i+blocksize,:]
        s += block[:,1].sum()

    sr = s/ds.shape[0]

    print(sr)
```

1004.2080517575215