


Python은 느리다!

파이썬은 single core, 즉 하나의 CPU만 사용하기 때문에 대용량 데이터를 다룰때 처리 속도가 느리다는 문제가 생긴다. 파이썬 자체는 싱글 코어에 램으로 작동하지만 파이썬의 몇몇 패키지는 멀티 코어를 사용할 수 있게 한다. 대표적이 예가 numpy 패키지로 이는 C 언어로 바인딩되어있기 때문에 multiple core, 또는 GPU를 사용할 수 있다. 이 포스팅에서는 파이썬에서 다중 코어를 사용해 데이터 병렬 처리를 하는 방법을 소개하려고 한다. 그 중에서도 대용량 데이터를 처리할 수 있는 `dask` 패키지를 소개한다.

Pandas는 데이터를 가공 패키지로 다양한 데이터 처리 함수를 제공해 데이터 가공에 흔히 사용되지만, 10GB가 넘어가는 데이터를 처리하는데는 속도가 느려진다는 단점이 있다. `dask`는 판다스보다 대용량 데이터를 더 빠르게 처리할 수 있으며, 컴퓨터의 램보다 더 큰 용량의 데이터도 로드할 수 있다.

dask란?

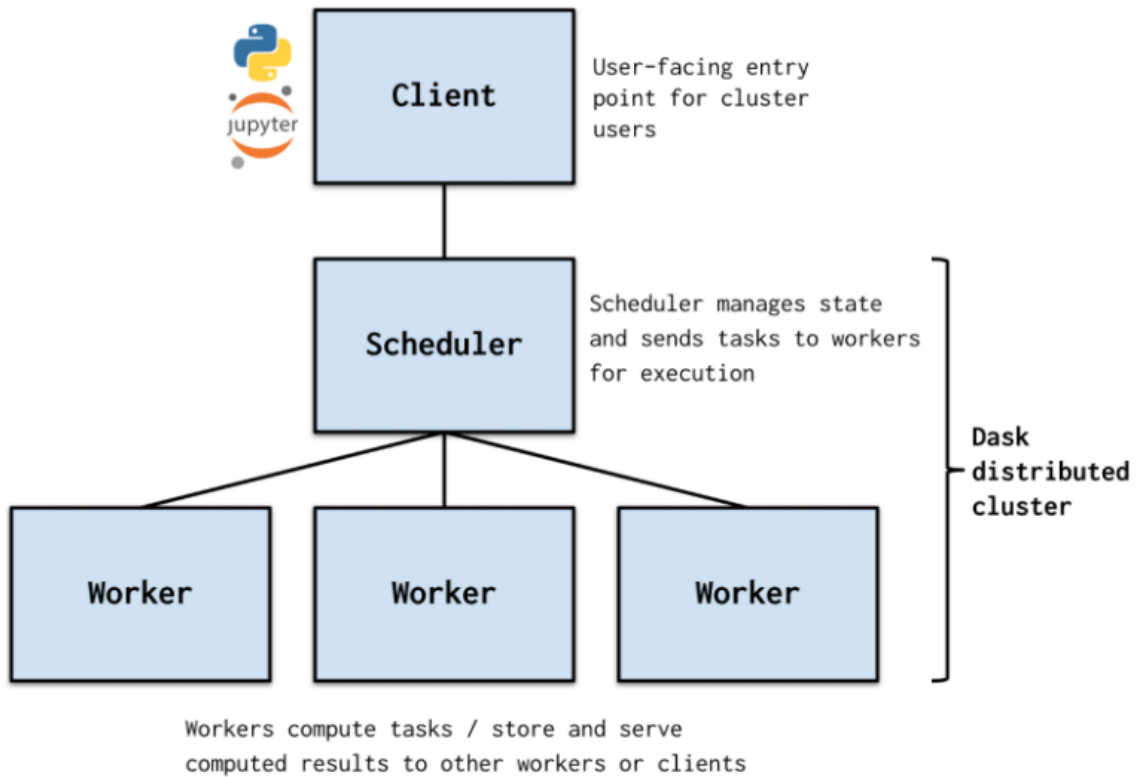
 dask는 python에서 멀티 코어로 병렬 처리를 가능하게 하는 패키지다. dask의 주요 특징은 다음과 같다.

- 작업 스케줄링 (dynamic task scheduling)
- 여러개의 코어를 사용하여 분산 처리
- 메모리 사이즈보다 큰 용량의 데이터도 처리 가능
- pandas, numpy와 사용 방법이 유사

dask cluster

dask는 dask cluster로 데이터를 병렬 처리한다.

cluster란 CPU, 메모리 관리 및 작업을 분배하는 역할을 하는 스케줄러(scheduler)인 Head 와 실제 작업을 수행하는 여러 Node 로 구성된다. 각 node들은 데이터 저장 공간(storage)을 공유하고 있다.



파이썬에서 데이터를 처리하는 사용자가 dask로 데이터 처리를 요청하면 (Client), Scheduler가 이 요청을 받아 여러 개의 하위 작업으로 나누어 worker에게 분배한다. Worker는 분할된 작업을 수행하고, 수행하는 과정에서 worker간 커뮤니케이션이 이루어진다.

dask schedulers

dask의 처리방식?

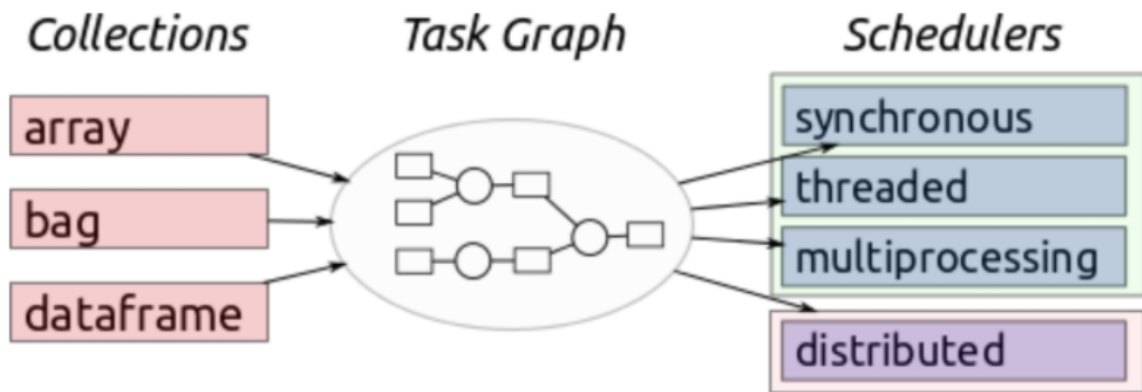
Dask는 작업 스케줄링(task scheduler)을 통해 task graph의 작업을 병렬로 처리한다.

- single machine scheduler

dask가 제공하는 default scheduler로, ~~single machine~~에서 스레드(thread)나 프로세스(process)를 사용해 병렬 처리를 수행함. 사용자가 직접 설정할 필요가 없다는 장점이 있음.

- distributed scheduler

비동기식(asynchronous) 병렬 처리 가능. 또한 처리 과정을 대시보드로 볼 수 있음. (port 8787)

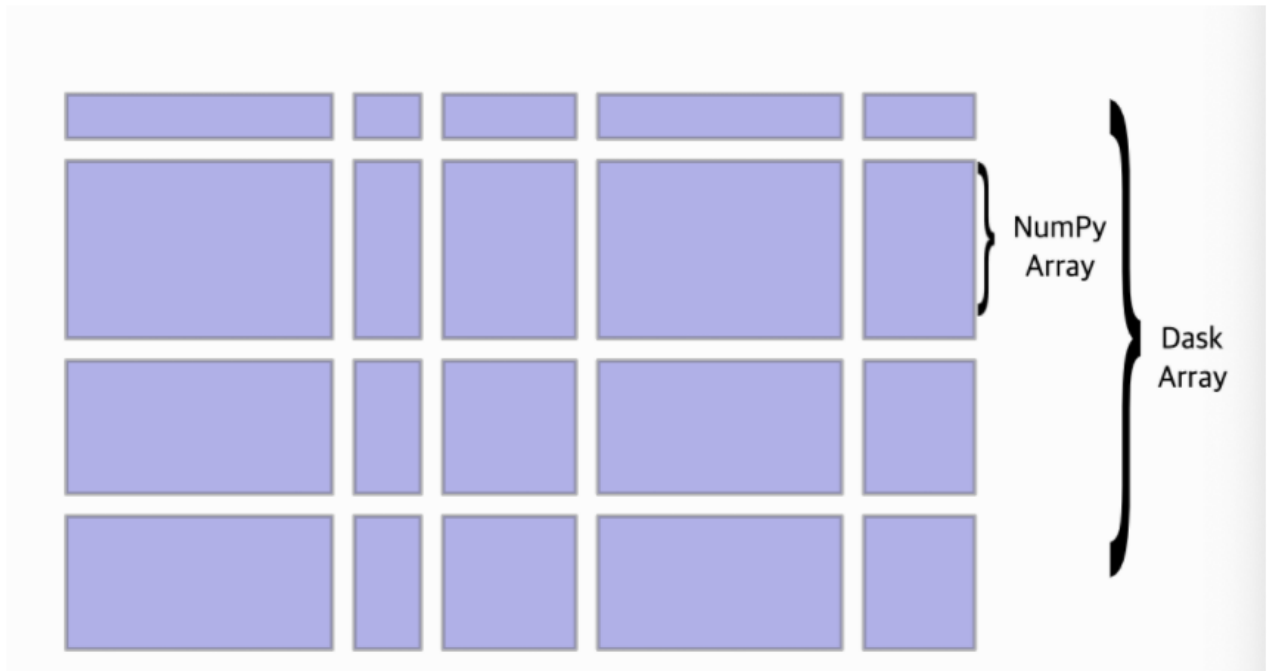


dask collections

dask의 자료구조?

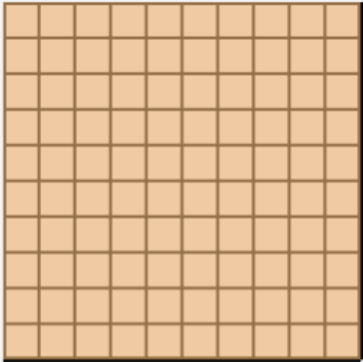
Dask는 Bag , Array , Dataframe collection을 제공하며 이는 각각 파이썬의 list, numpy, pandas와 유사하다. 하지만 dask의 collection은 대용량 데이터를 병렬 처리할 수 있다는 장점이 있다.

- dask collections: array



```
import dask.array as da

x = da.random.random((10000,10000), chunks=(1000,1000))
x
```

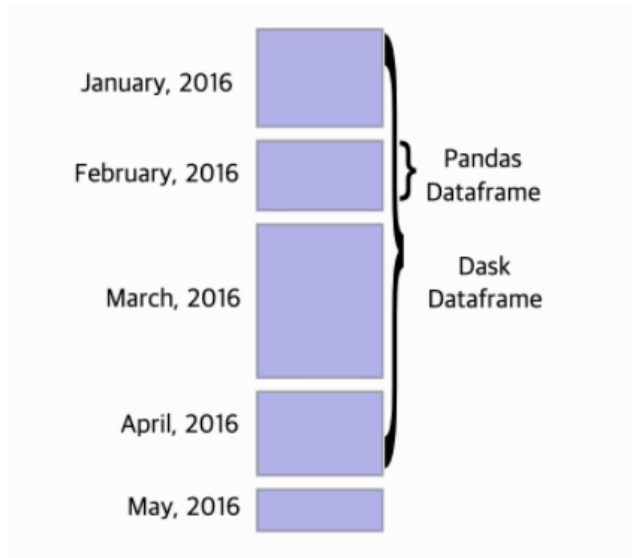
	Array	Chunk	
Bytes	800.00 MB	8.00 MB	
Shape	(10000, 10000)	(1000, 1000)	
Count	100 Tasks	100 Chunks	
Type	float64	numpy.ndarray	

dask array는 작은 numpy ndarray들의 모음이다. 10K10K의 매트릭스를 1K1k 청크 100개로 쪼갬다.

- **dask collections: dataframe**

```
import dask.dataframe as dd
```

dask.dataframe은 긴 row를 가진 데이터프레임을 처리하는데 효과적이다.



~~※~~ **dask.dataframe**은 pandas dataframe들이 row-wise하게 연결되어있으며, 실제 데이터프레임을 처리할 때는 row단위로 잘라서 병렬 처리한다. 따라서 pandas로 처리하기 힘든 긴(many rows) 데이터가 있다면, dask를 사용하자.

Tutorial: Advanced

참고

여기서는 1,300만 row를 가진 데이터프레임을 pandas와 dask로 불러와 처리 속도를 비교해보고자 한다.

- 데이터 로드

dask의 데이터 로드 속도는 pandas보다 1000배 빠르다.

이런 속도 차이가 가능한 이유는 dask는 데이터를 로드할 때 실제 연산을 수행하지 않기 때문이다. **Lazy Evaluation**은 어떤 값이 실제로 쓰일 때까지 계산을 미루는 동작 방식이다. dask는 lazy evaluation으로 작동하기 때문에 `compute()` 메소드를 호출하지 전까지 데이터가 메모리에 올라가지 않는다.

```
import pandas as pd
import numpy as np
import dask.dataframe as dd
import dask.bag as db
import json

timeit -r 2 -n 1 pd.read_csv("df4pid_r_date.tsv", sep = '\t')
# 17.4 s ± 134 ms per loop (mean ± std. dev. of 2 runs, 1 loop each)

timeit -r 2 -n 1 dd.read_csv("df4pid_r_date.tsv", sep = '\t')
# 15.9 ms ± 1.88 ms per loop (mean ± std. dev. of 2 runs, 1 loop each)
```

dask는 데이터를 로드할 때 (실제 값이 아닌) 그 구조만 가져온다.
데이터프레임은 아래 그림처럼 값이 아닌 자료형만 확인할 수 있다.

dd_df

Dask DataFrame Structure:

	rid	pid	r_date	keyterm	cat_1	cat_2	cat_3	cat_4	cat_5	is_sent_col	pos	neg	neu
npartitions=40													
	int64	int64	object	object	object	object	object	object	object	bool	int64	int64	int64

...

Dask Name: read-csv, 40 tasks

- 데이터 확인

dask는 데이터프레임의 뼈대만 로드하기 때문에 실제 값은 ^{'compute()' 메소드 호출시} 계산이 필요한 순간에 가져온다. 데이터를 확인하기 위해서도 df.head() 와 같이 계산식을 넣어줘야 한다.

```
timeit -r 5 -n 1 pd_df.head()
#149 µs ± 75.8 µs per loop (mean ± std. dev. of 5 runs, 1 loop each)
timeit -r 5 -n 1 dd_df.head()
#563 ms ± 28.3 ms per loop (mean ± std. dev. of 5 runs, 1 loop each)
```

결론적으로보면, 데이터프레임 계산은 pandas가 dask보다 더 빠르다고 할 수 있다. dataframe의 head만 보는데 밀리세컨드가 걸리는 것은 분석가 입장에서는 사용할 수 없는 수준의 속도이기 때문이다.

RAM 제한 사항 해결

병렬 컴퓨팅은 모두 좋습니다. 그러나 dask는 스토리지 문제를 어떻게 해결합니까? 음, 거대한 데이터 프레임을 여러 개의 작은 팬더 데이터 프레임으로 나눕니다. 그러나 더 작은 데이터 프레임의 누적 크기는 여전히 RAM보다 큽니다. 그렇다면 더 작은 데이터 프레임으로 나누면 문제가 어떻게 해결됩니까? 글쎄, dask는 이러한 작은 팬더 데이터 프레임을 로컬 저장소 (HDD / SSD)에 저장하고 필요할 때 개별 데이터 프레임의 데이터를 RAM으로 가져옵니다. 일반적인 PC의 로컬 저장 용량은 100GB이며 RAM 용량은 몇 GB로 제한됩니다. 따라서 dask를 사용하면 RAM 용량보다 훨씬 큰 데이터를 처리 할 수 있습니다.

예를 들어 데이터 프레임에 10 억 개의 행이 있다고 가정 해 보겠습니다. 이제 두 개의 열을 추가하여 세 번째 열을 생성 하려는 경우 팬더는 먼저 전체 데이터 프레임을 RAM에 로드 한 다음 계산을 수행합니다. 슬프게도 RAM의 크기에 따라 데이터로드 단계 자체에서 시스템을 압도 할 수 있습니다. *pandas의 방식* *error가 발생할 수 있음.* Dask는 데이터 프레임을 예를 들어 100 개의 청크로 분할합니다. 그런 다음 RAM에 1 개의 청크를 가져 와서 계산을 수행 한 다음 디스크로 다시 보냅니다. 다른 99 개의 청크로 이것을 반복합니다. 컴퓨터에 4 개의 코어가 있고 RAM이 4 개의 청크 크기와 동일한 데이터를 처리 할 수 있는 경우 모두 병렬로 작동하고 작업이 1/4의 시간에 완료됩니다. 가장 좋은 점은 관련된 코어 수나 RAM 용량에 대해 걱정할 필요가 없다는 것입니다. Dask는 백그라운드에서 모든 것을 파악하고 부담을주지 않습니다.

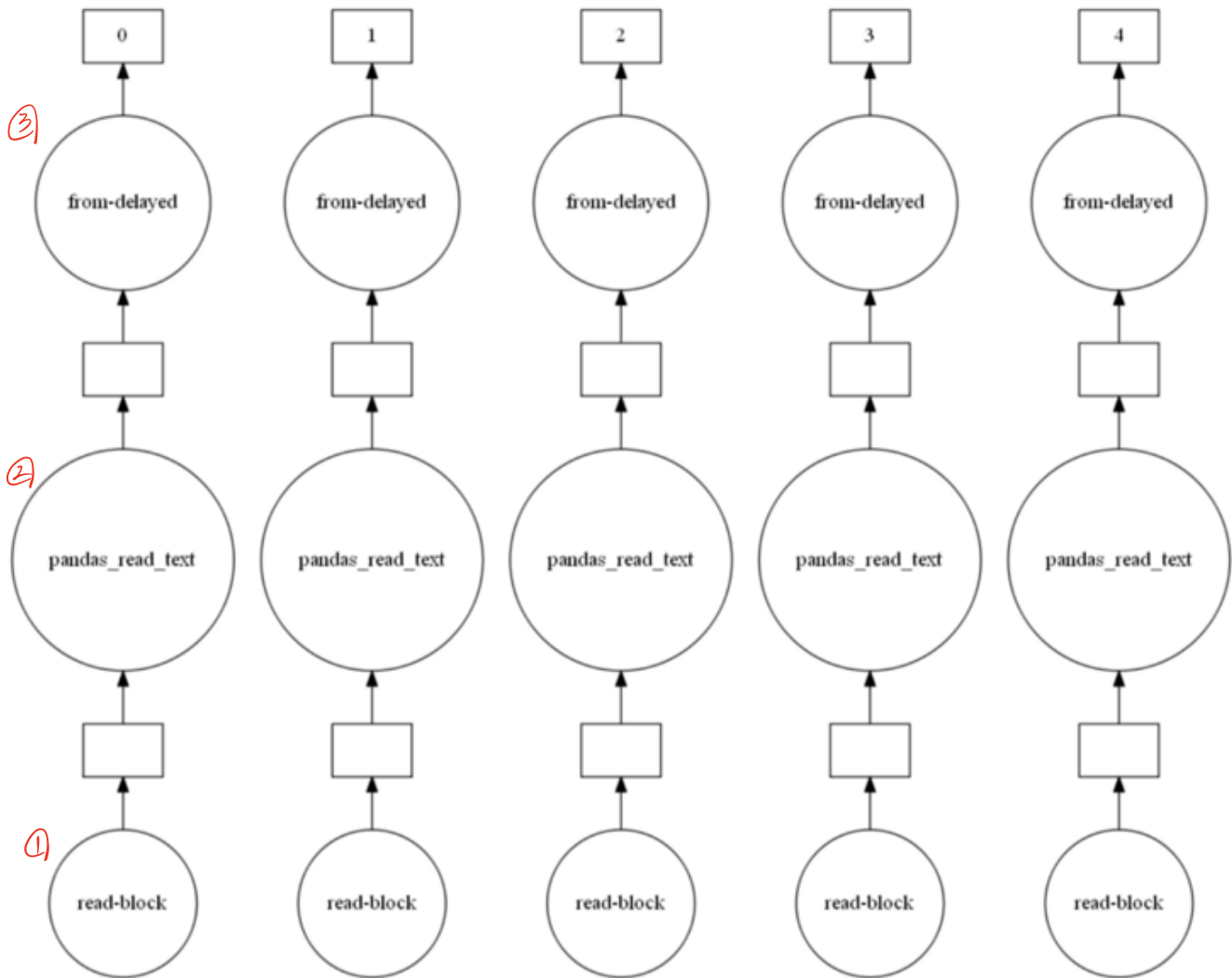
```
df_dd = dd.read_csv('data/lat_lon.csv',blocksize='600MB')
```

Dask DataFrame Structure:

	serial_no	latitude	longitude
npartitions=5			
	int64	float64	float64

...

Dask Name: from-delayed, 15 tasks



위의 이미지는 dask의 작업 그래프를 나타냅니다. 작업 그래프는 병렬 계산을 나타내는 dask의 방법입니다. 원은 작업 또는 기능을 나타내고 사각형은 출력 / 결과를 나타냅니다. 보시다시피 dask 데이터 프레임 만드는 프로세스에는 총 15 개의 작업 (파티션 당 3 개의 작업)이 필요합니다. 일반적으로 .NET과 같은 집계 계산을 수행하지 않는 한 dask 작업의 수는 파티션 수의 배수가 `max()` 됩니다. 첫 번째 단계에서는 600MB 블록을 읽습니다 (마지막 파티션의 경우 600MB 이하일 수 있음). 그런 다음 다음 단계에서이 바이트 블록을 pandas 데이터 프레임으로 변환합니다. 마지막 단계에서는 `from_delayed`를 사용하여 병렬 계산을 위해이 데이터 프레임을 준비합니다.

위 예시 한번 `task = 15` 일.

평신도 용어로 dask는 파이썬에서 병렬 컴퓨팅에 대한 인기있는 게이트웨이 중 하나입니다. 따라서 컴퓨터에 4 개의 코어가 있으면 계산을 위해 4 개의 코어를 모두 동시에 활용할 수 있습니다. numpy, pandas, scikit-learn 등과 같은 많은 인기있는 파이썬 라이브러리에 대한 dask 등가물이 있습니다. 이 게시물에서 우리는 pandas 등가를 인 dask 데이터 프레임에 관심이 있습니다.

Dask 데이터 프레임의 구조

7,400 만 개 이상의 위치 좌표가 포함 된 CSV 파일을 사용할 것입니다. 파일 크기는 약 2.5GB입니다. 컴퓨터에서 사용할 수 있는 RAM의 용량에 따라 팬더를 사용하는 시스템을 압도 할 수도 있고 그렇지 않을 수도 있습니다. 12GB RAM이 있는 쿼드 코어 머신을 사용할 것입니다. 그래서 팬더는 내 컴퓨터에서이 데이터 프레임을 편안하게 처리 할 수 있었습니까? 이것은 나중에 pandas와 dask를 벤치마킹 할 때 유용 할 것이며 dask를 언제 사용하고 언제 Pandas를 사용해야 하는지 이해하는 데 도움이 될 것입니다.

CSV를 읽어 보겠습니다.

```
import dask.dataframe as dd
df_dd = dd.read_csv('data/lat_lon.csv')
```

In [12]: df_dd

Out[12]: **Dask DataFrame Structure:**

	serial_no	latitude	longitude
npartitions=41			
	int64	float64	float64

...

Dask Name: from-delayed, 123 tasks

데이터 프레임 출력 완료

보시다시피 pandas와 달리 여기에서는 (실제 데이터 프레임이 아닌) 데이터 프레임의 구조를 얻습니다. 생각해 보면 말이 됩니다. Dask는 (RAM이 아닌) 디스크에 있는 여러 청크로 데이터 프레임을 로드했습니다. 데이터 프레임을 출력해야 하는 경우 먼저 모든 데이터를 RAM으로 가져 와서 함께 연결 한 다음 최종 데이터 프레임을 보여 주어야 합니다. *을 사용하여 강제로 수행 할 때까지 그렇게 하지 않습니다. `.compute()` . 나중에 더 자세히 설명하겠습니다.

구조 자체는 많은 정보를 전달합니다. ① 예를 들어, 데이터 프레임이 41 개의 청크로 분할되었음을 보여줍니다. ② 데이터 프레임의 열과 해당 데이터 유형을 보여줍니다. 여태까지는 그런대로 잘됐다. 그러나 잠깐만 요, dask 이름과 작업 수를 포함하는 마지막 줄의 의미는 무엇입니까? 이 작업은 무엇입니까? 이름의 의미는 무엇입니까?

41 x 3 = 123

Dask의 작업 그래프

`.visualize()` 메서드를 호출하여 이러한 답변을 많이 얻을 수 있습니다. 시도해 봅시다. 이 방법이 작동 하려면 `Graphviz` 라이브러리 가 필요합니다.

```
df_dd.visualize()
```

