Nama : Muh Ibnu Fajar

Nim : 235510015

Prodi : Teknik Komputer S1

BASIC EXAMPLES

1. Array.py

Programnya:

```
import dask.array as da

x = da.random.random((1000, 1000), chunks=(100, 100))
y = x.mean()
print("Dask mean result:", y.compute())

from dask.distributed import Client
import dask.array as da

def main():
    client = Client(processes=False, threads_per_worker=4, n_workers=1, memory_limit='2GB')
    print("Dask client:", client)

    x = da.random.random((10000, 10000), chunks=(1000, 1000))
    print("Dask array x:", x)

    y = x + x.T
    z = y[::2, 5000:].mean(axis=1)
    print("Dask array z (lazy):", z)

    result = z.compute()
    print("Result (numpy array):", result)

    y_persisted = y.persist()
    print("y persisted:", y_persisted)

    first_elem = y_persisted[0, 0].compute()
    total_sum = y_persisted.sum().compute()
    print("First element y[0,0]:", first_elem)
    print("Sum of y:", total_sum)

    client.close()
```

```
if '__name__' == '__main__':
    main()
```

Outputnya:

```
Dask mean result: 0.5002806884552752
PS C:\xampp\htdocs\Computasi cloud>
```

2. Bags.py

Programnya:

```
import dask.bag as db
import json

def main():

    data = [
        {"name": "Alice", "age": 25, "occupation": "Engineer"},
        {"name": "Bob", "age": 40, "occupation": "Doctor"},
        {"name": "Charlie", "age": 35, "occupation": "Engineer"},
        {"name": "Diana", "age": 28, "occupation": "Teacher"},
        {"name": "Eve", "age": 45, "occupation": "Doctor"},
    ]

    with open("people.json", "w", encoding="utf-8") as f:
        for rec in data:
            f.write(json.dumps(rec) + "\n")

    b = db.read_text("people.json").map(json.loads)

    print("All records:", b.compute())

    b2 = b.filter(lambda rec: rec.get("age", 0) > 30)
    print("Filtered (age > 30):", b2.compute())

    occup = b2.map(lambda rec: rec.get("occupation"))
    print("Occupations:", occup.compute())

    freq = occup.frequencies()
    print("Frequencies:", freq.compute())

    top = freq.topk(5, key=lambda x: x[1])
    print("Top occupations:", top.compute())
```

```
if __name__ == "__main__":
    main()
```

Outputnya:

```
"c:/xampp/htdocs/Computasi cloud/pertemuan 8/basic_examples/bags.py"
All records: [{'name': 'Alice', 'age': 25, 'occupation': 'Engineer'}, {'name': 'Bob', 'age': 40,
'occupation': 'Doctor'}, {'name': 'Charlie', 'age': 35, 'occupation': 'Engineer'}, {'name': 'Diana',
'age': 28, 'occupation': 'Teacher'}, {'name': 'Eve', 'age': 45, 'occupation': 'Doctor'}]
Filtered (age > 30): [{'name': 'Bob', 'age': 40, 'occupation': 'Doctor'}, {'name': 'Charlie', 'age':
35, 'occupation': 'Engineer'}, {'name': 'Eve', 'age': 45, 'occupation': 'Doctor'}]
Occupations: ['Doctor', 'Engineer', 'Doctor']
Frequencies: [('Doctor', 2), ('Engineer', 1)]
Top occupations: [('Doctor', 2), ('Engineer', 1)]
```

3. DataFrames.py

Programnya:

```
from dask.distributed import Client
import dask
import dask.dataframe as dd

def main():
    client = Client(n_workers=2, threads_per_worker=2, memory_limit="1GB")
    print(client)

    df = dask.datasets.timeseries()

    print("Dtypes:\n", df.dtypes)
    print(df.head(3))

    df2 = df[df.y > 0]
    df3 = df2.groupby("name").x.std()
    print("Series (belum dihitung):", df3)

    computed = df3.compute()
    print("Hasil compute():\n", computed)

    df4 = df.groupby("name").aggregate({"x": "sum", "y": "max"})
    res4 = df4.compute()
    print("Hasil aggregasi sum x, max y per name:\n", res4)

    df4_small = df4.repartition(npartitions=1)
```

```
    joined = df.merge(df4_small, left_on="name", right_index=True,
                      suffixes=("_original", "_aggregated"))
    print(joined.head())

    df = df.persist()
    print("Persisted:", df)

    resampled = df[["x", "y"]].resample("1h").mean()
    print(resampled.head())

if __name__ == "__main__":
    main()
```

Outputnya:

```

```

4. Delayed_example.py

Programnya:

```
from dask import delayed
import time

def inc(x):
    time.sleep(1)
    return x + 1

def double(x):
    time.sleep(1)
    return x * 2

def add(x, y):
    time.sleep(1)
    return x + y

if __name__ == "__main__":

    a = delayed(inc)(10)
    b = delayed(double)(a)
    c = delayed(add)(a, b)

    print("Computing...")
    result = c.compute()
```

```
    print("Result:", result)
```

Outputnya:

```
"c:/xampp/htdocs/Computasi cloud/pertemuan 8/basic_examples/delayed_example.py"
Computing...
Result: 33
PS C:\xampp\htdocs\Computasi cloud>
```

5. Futures_example.py

Programnya:

```
from dask.distributed import Client, as_completed
import time
import random

def slow_increment(x):
    time.sleep(random.uniform(0.5, 1.5))
    return x + 1

def slow_add(x, y):
    time.sleep(random.uniform(0.5, 1.5))
    return x + y

def main():
    client = Client(n_workers=2, threads_per_worker=1)
    print("Client:", client)

    futures = []
    for i in range(5):
        fut = client.submit(slow_increment, i)
        futures.append(fut)

    results = client.gather(futures)
    print("Results of slow_increment:", results)

    fut2 = client.submit(slow_add, results[0], results[1])
    print("Result of slow_add:", fut2.result())

    print("Iterating as futures complete:")
    for completed in as_completed(futures):
        res = completed.result()
        print(" - completed:", res)
```

```
    client.close()

if __name__ == "__main__":
    main()
```

Outputnya:

```
"c:/xampp/htdocs/Computasi cloud/pertemuan 8/basic_examples/futures_example.py"
Client: <Client: 'tcp://127.0.0.1:64826' processes=2 threads=2, memory=15.41 GiB>
Results of slow_increment: [1, 2, 3, 4, 5]
Result of slow_add: 3
Iterating as futures complete:
 - completed: 1
 - completed: 2
 - completed: 3
 - completed: 4
 - completed: 5
```

6. Ml_example.py

Programnya:

```
from dask_ml.datasets import make_classification
from dask_ml.model_selection import train_test_split
from dask_ml.linear_model import LogisticRegression
from dask.distributed import Client
from sklearn.metrics import accuracy_score

if __name__ == "__main__":
    client = Client(processes=False)

    print("Generating dataset...")
    X, y = make_classification(
        n_samples=20000,
        n_features=20,
        chunks=2000
    )

    print("Splitting data...")
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.3
    )

    print("Training Logistic Regression...")
    model = LogisticRegression(solver="lbfgs")
```

```
    model.fit(X_train, y_train)

    print("Predicting...")
    y_pred = model.predict(X_test)

    y_test = y_test.compute()
    y_pred = y_pred.compute()

    acc = accuracy_score(y_test, y_pred)
    print("Accuracy:", acc)
```

Outputnya:

```

```

7. xarray_dask_example

Programnya:

```
import xarray as xr
import dask.array as da
import pandas as pd

def main():
    times = pd.date_range("2000-01-01", periods=365, freq="D")
    x = range(100)
    y = range(50)

    data = da.random.random(
        size=(len(times), len(x), len(y)),
        chunks=(50, 100, 50)
    )

    ds = xr.Dataset(
        {"var": (("time", "x", "y"), data)},
        coords={
            "time": times,
            "x": x,
            "y": y
        }
    )

    print("Dataset XArray (lazy, dask-backed):")
    print(ds)
```

```
    mean_time = ds["var"].mean(dim=("x", "y"))

    print("\nLazy mean over x & y (per time):")
    print(mean_time)

    result = mean_time.compute()
    print("\nHasil compute():")
    print(result)

    result.to_netcdf("mean_time.nc")
    print("\nHasil disimpan ke mean_time.nc")

if __name__ == "__main__":
    main()
```

Outputnya:

```
"c:/xampp/htdocs/Computasi cloud/pertemuan 8/basic_examples/xarray_dask_example.py"
Dataset XArray (lazy, dask-backed):
<xarray.Dataset> Size: 15MB
Dimensions:  (time: 365, x: 100, y: 50)
Coordinates:
  * time     (time) datetime64[ns] 3kB 2000-01-01 2000-01-02 ... 2000-12-30
  * x        (x) int64 800B 0 1 2 3 4 5 6 7 8 9 ... 91 92 93 94 95 96 97 98 99
  * y        (y) int64 400B 0 1 2 3 4 5 6 7 8 9 ... 41 42 43 44 45 46 47 48 49
Data variables:
    var      (time, x, y) float64 15MB dask.array<chunksize=(50, 100, 50), meta=np.ndarray>

Lazy mean over x & y (per time):
<xarray.DataArray 'var' (time: 365)> Size: 3kB
dask.array<mean_agg-aggregate, shape=(365,), dtype=float64, chunksize=(50,),
chunktype=numpy.ndarray>
Coordinates:
  * time     (time) datetime64[ns] 3kB 2000-01-01 2000-01-02 ... 2000-12-30

Hasil compute():
<xarray.DataArray 'var' (time: 365)> Size: 3kB
array([0.50001681, 0.49725142, 0.50329651, 0.50599896, 0.5027561 ,
       0.49894099, 0.4997813 , 0.49468632, 0.50267906, 0.50031155,
       0.50364879, 0.50239821, 0.4971227 , 0.50239318, 0.50432016,
       0.4954109 , 0.49805459, 0.49954872, 0.50256413, 0.49679546,
       0.49564205, 0.50126087, 0.49761468, 0.49329746, 0.50473385,
       0.50762099, 0.50446   , 0.50225543, 0.49750337, 0.49747673,
       0.4963567 , 0.50474453, 0.50313544, 0.497189  , 0.50201546,
       0.50369161, 0.49943136, 0.49816137, 0.49287228, 0.50100644,
```

```
      0.49668706, 0.5093561 , 0.49969619, 0.49723864, 0.49169536,
      0.49978981, 0.50128714, 0.49750297, 0.50596049, 0.49534614,
      0.50746989, 0.4977113 , 0.49706951, 0.50033977, 0.50508106,
      0.49653489, 0.49275202, 0.49737201, 0.49527558, 0.50224366,
      0.50121546, 0.50031054, 0.49799911, 0.49550964, 0.49831111,
      0.5044226 , 0.49480005, 0.50575642, 0.49549111, 0.50322942,
      0.50805483, 0.50138332, 0.50260014, 0.50046285, 0.48975797,
      0.50184976, 0.50358165, 0.49925086, 0.49798923, 0.49708552,
      0.50808597, 0.50109234, 0.49706769, 0.49263649, 0.50229542,
      0.49946067, 0.49895568, 0.5041667 , 0.50152456, 0.4983973 ,
      0.5018171 , 0.49479274, 0.5028548 , 0.50664327, 0.50194677,
      0.49486593, 0.50016075, 0.50195203, 0.49119574, 0.50796076,
...
      0.49958063, 0.50300148, 0.50027401, 0.49807323, 0.49613214,
      0.50041508, 0.51152182, 0.50450138, 0.49907704, 0.49760603,
      0.49639227, 0.50451681, 0.50906984, 0.50364188, 0.50241142,
      0.49958427, 0.4945443 , 0.50197579, 0.49696853, 0.50089659,
      0.50000712, 0.50198974, 0.49848277, 0.49975784, 0.49498833,
      0.50082468, 0.49585263, 0.50467452, 0.49670163, 0.49956506,
      0.49881657, 0.50348377, 0.49357166, 0.49817658, 0.49548422,
      0.49921355, 0.49696108, 0.49807257, 0.49691806, 0.50201093,
      0.50147796, 0.5002794 , 0.49570815, 0.50404074, 0.5025808 ,
      0.49944395, 0.49514195, 0.49105436, 0.49674017, 0.49564757,
      0.49778784, 0.49715674, 0.5014883 , 0.50046003, 0.50430951,
      0.50112908, 0.4950811 , 0.49826338, 0.49911712, 0.49964819,
      0.50807491, 0.49536646, 0.50439224, 0.50643001, 0.49531784,
      0.50880324, 0.5001901 , 0.50366725, 0.49877876, 0.49896895,
      0.50672266, 0.50427271, 0.49942921, 0.50393323, 0.50748265,
      0.49970736, 0.50091663, 0.50548505, 0.49453395, 0.49582626,
      0.50413201, 0.50303985, 0.50326471, 0.49975771, 0.50196151,
      0.50453802, 0.49833353, 0.50117533, 0.49895628, 0.50297753,
      0.50223532, 0.48827916, 0.4950841 , 0.50103674, 0.49368212,
      0.49455562, 0.49879112, 0.49337725, 0.50883076, 0.50912309])
Coordinates:
  * time     (time) datetime64[ns] 3kB 2000-01-01 2000-01-02 ... 2000-12-30

Hasil disimpan ke mean_time.nc
```

DATAFRAMES

1. Dask_groupby.py

Programnya:

Outputnya:

APPLICATIONS

1. json_dask_web.py

Programnya:

```
import json
import requests
from dask import delayed, compute

@delayed
def fetch(url):
    print(f"Mengambil data dari: {url}")
    r = requests.get(url)
    return r.json()

urls = [
    "https://jsonplaceholder.typicode.com/todos/1",
    "https://jsonplaceholder.typicode.com/todos/2",
    "https://jsonplaceholder.typicode.com/todos/3",
]

tasks = [fetch(u) for u in urls]

results = compute(*tasks)

print("\nHasil JSON yang diterima:")
for r in results:
    print(r)
```

Outputnya:

```
(venv) PS C:\xampp\htdocs\Computasi cloud\pertemuan 8> python
.\Applications\json_dask_web.py
Mengambil data dari: https://jsonplaceholder.typicode.com/todos/2
Mengambil data dari: https://jsonplaceholder.typicode.com/todos/1
Mengambil data dari: https://jsonplaceholder.typicode.com/todos/3

Hasil JSON yang diterima:
{'userId': 1, 'id': 1, 'title': 'delectus aut autem', 'completed': False}
```

```
{'userId': 1, 'id': 2, 'title': 'quis ut nam facilis et officia qui', 'completed': False}
{'userId': 1, 'id': 3, 'title': 'fugiat veniam minus', 'completed': False}
```

2. async_dask_example.py

Programnya:

```
import asyncio
from dask.distributed import Client
from dask import delayed

def square(x):
    return x * x

@delayed
def delayed_square(x):
    return square(x)

async def main():
    print("Dask Cluster running...")
    client = Client(processes=True, n_workers=2, threads_per_worker=1)
    print(client)

    print("Menghitung secara asynchronous ...")

    tasks = [delayed_square(i) for i in range(10)]

    loop = asyncio.get_running_loop()

    result = await loop.run_in_executor(
        None,
        lambda: client.gather(client.compute(tasks))
    )

    print("Hasil perhitungan:", result)

    client.close()

if __name__ == "__main__":
    asyncio.run(main())
```

Outputnya:

```
python async_dask_example.py
Dask Cluster running...
```

```
<Client: 'tcp://127.0.0.1:64931' processes=2 threads=2, memory=15.41 GiB>
Menghitung secara asynchronous ...
Hasil perhitungan: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

3. async_web_server

Programnya:

```
from aiohttp import web
from dask.distributed import Client
import asyncio
def heavy_computation(x):
    import time
    time.sleep(2)
    return x * x

async def handle(request):
    n = int(request.query.get('n', '2'))
    future = client.submit(heavy_computation, n)
    result = await asyncio.wrap_future(future)
    return web.Response(text=f"Result: {result}")

async def init_app():
    app = web.Application()
    app.router.add_get('/', handle)
    return app

if __name__ == "__main__":
    client = Client()
    print("Dask Cluster running…", client)
    app = asyncio.run(init_app())
    web.run_app(app, port=8080)
```

Outputnya:

```
cd "C:\xampp\htdocs\Computasi cloud\pertemuan 8\Applications"
>> python async_web_server.py
>>
Dask Cluster running… <Client: 'tcp://127.0.0.1:57007' processes=4 threads=12,
memory=15.41 GiB>
======== Running on http://0.0.0.0:8080 ========
```

```
(Press CTRL+C to quit)
```

4. dask_parallel_example.py.

Programnya:

```
from dask.distributed import Client
import time
import random

def slow_task(x):
    time.sleep(random.uniform(0.5, 1.5))
    return x * x

def main():
    client = Client()
    print("Dask Cluster:", client)

    inputs = list(range(10))
    print("Inputs:", inputs)

    futures = [client.submit(slow_task, x) for x in inputs]

    results = client.gather(futures)

    print("Results:", results)

    client.close()

if __name__ == "__main__":
    main()
```

Outputnya:

```
python dask_parallel_example.py
Dask Cluster: <Client: 'tcp://127.0.0.1:55507' processes=4 threads=12, memory=15.41 GiB>
Inputs: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Results: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

5. evolving_workflow_example.py

Programnya:

```
from dask.distributed import Client, as_completed
import time
import random

def task(x):
    time.sleep(random.uniform(0.5, 1.5))
    return x * x

def main():
    client = Client()
    print("Dask cluster:", client)

    inputs = list(range(5))
    futures = [client.submit(task, i) for i in inputs]

    results = []
    for future in as_completed(futures):
        res = future.result()
        print("Done:", res)
        results.append(res)

        if res < 10:
            new_future = client.submit(task, res + 10)
            futures.append(new_future)

    print("All results:", results)

    client.close()

if __name__ == "__main__":
    main()
```

Outputnya:

```
python evolving_workflow_example.py
Dask cluster: <Client: 'tcp://127.0.0.1:53589' processes=4 threads=12, memory=15.41 GiB>
Done: 0
Done: 1
Done: 9
Done: 4
Done: 16
All results: [0, 1, 9, 4, 16]
```

6. dask_image_processing.py

Programnya:

```
import os
from dask import delayed, compute
from PIL import Image

def process_image(path, out_folder, size=(256, 256)):
    img = Image.open(path)
    img = img.resize(size)
    arr = img.convert("L")
    basename = os.path.basename(path)
    name, ext = os.path.splitext(basename)
    out_path = os.path.join(out_folder, name + "_gray.png")
    arr.save(out_path)
    return out_path

def main():
    input_folder = "images"
    output_folder = "processed"
    os.makedirs(output_folder, exist_ok=True)

    if not os.path.isdir(input_folder):
        print("Folder input tidak ditemukan:", input_folder)
        return

    paths = [os.path.join(input_folder, f)
             for f in os.listdir(input_folder)
             if f.lower().endswith((".png", ".jpg", ".jpeg", ".bmp"))]

    if not paths:
        print("Tidak ada gambar di folder:", input_folder)
        return

    print("Found images:", paths)

    tasks = [delayed(process_image)(p, output_folder) for p in paths]

    results = compute(*tasks)
    print("Processed images:", results)

if __name__ == "__main__":
    main()
```

Outputnya:

```
python dask_image_processing.py
Found images: ['images\\foto1.png']
Processed images: ('processed\\foto1_gray.png',)
```

7. tujuh
8. dask_numba_stencil.py

Programnya:

```
import dask.array as da
from numba import jit
import numpy as np

@jit(nopython=True)
def stencil_step(a):
    n, m = a.shape
    b = np.empty_like(a)
    for i in range(1, n-1):
        for j in range(1, m-1):
            b[i, j] = 0.2*(a[i, j] + a[i-1,j] + a[i+1,j] + a[i,j-1] + a[i,j+1])
    return b

def main():
    x = da.random.random((1000, 1000), chunks=(250, 250))

    y = x.map_blocks(lambda block: stencil_step(block), dtype=float)

    result = y.compute()

    print("Processed array shape:", result.shape)
    print("Contoh nilai di tengah:", result[result.shape[0]//2, result.shape[1]//2])

if __name__ == "__main__":
    main()
```

Outputnya:

```
python dask_numba_stencil.py
Processed array shape: (1000, 1000)
Contoh nilai di tengah: 0.0
```

9. dask_prophet_example.py

programnya:

```
import pandas as pd
import dask.dataframe as dd
from dask.distributed import Client
from prophet import Prophet
import matplotlib.pyplot as plt

def main():
    client = Client()
    print("Dask Client:", client)

    dates = pd.date_range(start='2020-01-01', periods=365, freq='D')
    y = (pd.Series(10 + 5 * np.sin(2 * np.pi * dates.dayofyear / 365) +
                   np.random.randn(len(dates)), index=dates)).values

    df = pd.DataFrame({'ds': dates, 'y': y})
    print("Sample data (pandas):")
    print(df.head())

    ddf = dd.from_pandas(df, npartitions=4)
    df_clean = ddf.compute()

    m = Prophet()
    m.fit(df_clean)

    future = m.make_future_dataframe(periods=30)
    forecast = m.predict(future)

    print("\nForecast head:")
    print(forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].head())

    fig = m.plot(forecast)
    plt.show()

if __name__ == "__main__":
    import numpy as np
    main()
```

outputnya:

```
Sample data (pandas):
```

```
        ds         y
0 2020-01-01   9.459088
1 2020-01-02  10.121898
2 2020-01-03  10.590657
3 2020-01-04   8.221116
4 2020-01-05  10.462601
08:03:24 - cmdstanpy - INFO - Chain [1] start processing
08:03:26 - cmdstanpy - INFO - Chain [1] done processing

Forecast head:
        ds      yhat  yhat_lower  yhat_upper
0 2020-01-01  10.827872    9.490836   12.166955
1 2020-01-02  10.956484    9.670758   12.323625
2 2020-01-03  11.258409   10.057082   12.506888
3 2020-01-04  11.074593    9.866299   12.277780
4 2020-01-05  11.203149    9.935828   12.442759
```