MODUL 5
Komunikasi Sistem Paralel

1.1. Tujuan

a. Memahami konsep komunikasi antar proses dalam sistem paralel.
b. Mengimplementasikan Message Passing Interface (MPI) menggunakan mpi4py pada Python.
c. Menggunakan operasi komunikasi:
    a. Broadcast untuk mengirim data ke seluruh proses.
    b. Scatter untuk membagi data ke beberapa proses.
    c. Gather untuk mengumpulkan hasil kembali ke satu proses (root).
d. Menganalisis kinerja komunikasi dalam konteks komputasi paralel.

1.2. Alat & Bahan

1. Perangkat Lunak:
    - Python, mpi4py (pip install mpi4py)
    - OpenMPI (diinstal di sistem)
2. Perangkat Keras:
    - Laptop/PC dengan RAM minimal 8 GB
    - Koneski internet

1.3. Materi

1. Komunikasi dalam Sistem Paralel
   Menurut Fujimoto (2000) dalam Parallel and Distributed Simulation Systems, komunikasi dalam sistem paralel terjadi melalui pertukaran pesan antar proses yang berjalan secara bersamaan pada node berbeda. Tujuan utamanya adalah sinkronisasi data dan pembagian beban kerja.
2. Konsep Message Passing
   Message Passing Interface (MPI) merupakan standar komunikasi antar proses dalam sistem paralel (Taniar et al., 2008). Komunikasi dapat bersifat Synchronous yaitu proses pengirim menunggu sampai penerima menerima pesan dan Asynchronous yaitu proses pengirim tidak perlu menunggu penerima selesai.
3. Komunikasi pada Jaringan
   Menurut Davies (2019) dalam Networking Fundamentals, komunikasi antar node dalam jaringan bergantung pada protokol transmisi (misalnya TCP/IP) yang memastikan pesan

sampai ke tujuan dengan benar. Dalam sistem paralel, komunikasi antar proses atau node bisa terjadi dalam beberapa pola tergantung bagaimana pesan dikirim dari satu proses ke proses lain.

1. Unicast adalah proses komunikasi dimana satu proses mengirim pesan ke satu proses tujuan saja. Contoh Proses P1 mengirimkan hasil komputasi ke P2. Di jaringan komputer unicast seperti komunikasi client-server biasa (misalnya HTTP request dari browser ke satu server).

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = "Halo dari proses 0"
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
    print(f"Proses 1 menerima pesan: {data}")
```

2. Broadcast adalah komunikasi dimana satu proses mengirim pesan yang sama ke semua proses lain dalam satu grup. Contoh Proses master P0 mengirim parameter model ke seluruh worker P1, P2, P3. Biasa digunakan dalam distributed training (deep learning) atau simulasi paralel.

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = "Parameter awal model"
else:
    data = None

data = comm.bcast(data, root=0)
print(f"Proses {rank} menerima: {data}")
```

3. Scatter dan Gather adalah komunikasi yang digunakan untuk mendistribusikan data besar ke banyak proses, lalu menggabungkan kembali hasilnya. Scatter artinya data besar dipecah dan dikirim ke tiap proses dan Gather artinya hasil dari tiap proses dikumpulkan ke root. Contoh kasus File data besar dibagi 4 bagian ke 4 node paralel (scatter). Setelah dihitung rata-rata tiap bagian, hasilnya dikumpulkan kembali (gather).

```python
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

data_per_process = 4
total_data = data_per_process * size

if rank == 0:
    data = np.arange(total_data, dtype=int)
    print(f"Data awal di proses 0: {data}")
else:
    data = None

local_data = np.empty(data_per_process, dtype=int)

# SCATTER: bagi data ke tiap proses
comm.Scatter(data, local_data, root=0)
print(f"Proses {rank} menerima data: {local_data}")

# Setiap proses menghitung hasil parsial
local_result = local_data ** 2
print(f"Proses {rank} hasil parsial: {local_result}")

# GATHER: kumpulkan hasil ke proses 0
if rank == 0:
    final_result = np.empty(total_data, dtype=int)
else:
    final_result = None

comm.Gather(local_result, final_result, root=0)

# Cetak hasil akhir
if rank == 0:
    print(f"\nHasil akhir setelah Gather: {final_result}")
```

4. Multicast adalah komunikasi dimana satu proses mengirim pesan hanya ke sekelompok proses tertentu (tidak ke semua). Contoh Proses P1 mengirim pesan hanya ke P2 dan P3 dalam subkelompok worker. Biasa digunakan dalam sistem besar di mana tidak semua node perlu tahu semua data. Dalam MPI, multicast dapat disimulasikan dengan communicator subset, misalnya MPI.Comm.Split() untuk membuat grup baru.

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Pembentukan grup multicast

color = 1 if rank < 3 else MPI.UNDEFINED
sub_comm = comm.Split(color, rank)

#Pengiriman pesan multicast
if rank == 0:
    data = "Pesan penting hanya untuk grup tertentu"
    print(f"Proses {rank} mengirim: '{data}' ke grup multicast (0,1,2)")

if sub_comm:

    data = sub_comm.bcast(data if rank == 0 else None, root=0)
    print(f"Proses {rank} menerima pesan multicast: {data}")
else:
    print(f"Proses {rank} tidak termasuk grup multicast.")
```

1.4.Langkah Praktikum

1. Buat Server

Berikut merupakan potongan kode untuk Server

```python
import socket
import threading

HOST = '127.0.0.1'
PORT = 5000

# Menyimpan daftar koneksi client
clients = []

# Fungsi untuk menangani pesan dari setiap client
def handle_client(conn, addr):
    print(f"[TERHUBUNG] Client baru dari {addr}")
    while True:
        try:
            data = conn.recv(1024).decode('utf-8')
            if not data:
                break
            print(f"[PESAN dari {addr}]: {data}")

            # Kirim pesan ke semua client lain (broadcast)
            broadcast(f"[{addr}] {data}", conn)
        except:
            break
    print(f"[PUTUS] Koneksi dari {addr}")
    clients.remove(conn)
    conn.close()

# Fungsi broadcast
def broadcast(message, sender_conn):
    for client in clients:
        if client != sender_conn:
            try:
                client.sendall(message.encode('utf-8'))
            except:
                client.close()
                clients.remove(client)
```

```python
# Main server
def main():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((HOST, PORT))
    server.listen()
    print(f"[SERVER AKTIF] Menunggu koneksi di {HOST}:{PORT}")

    while True:
        conn, addr = server.accept()
        clients.append(conn)
        thread = threading.Thread(target=handle_client, args=(conn, addr))
        thread.start()

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

2. Buat Client
   Berikut potongan code client

```python
import socket
import threading

HOST = '127.0.0.1'
PORT = 5000

# Terima pesan dari server
def receive_messages(sock):
    while True:
        try:
            msg = sock.recv(1024).decode('utf-8')
            if not msg:
                break
            print(f"\n📢 Broadcast: {msg}")
        except:
            print("Koneksi ke server terputus.")
            break

# Main client
def main():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((HOST, PORT))
    print("Terhubung ke server. Ketik pesan dan tekan Enter untuk mengirim.")


    threading.Thread(target=receive_messages, args=(client,), daemon=True).start()

    while True:
        msg = input("Kamu: ")
        if msg.lower() == 'exit':
            break
        client.sendall(msg.encode('utf-8'))

    client.close()

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

3. Implementasi Broadcast antara Server dan Client Berikut potongan code disisi Server.

```python
import socket
import threading
from mpi4py import MPI
import numpy as np

HOST = '127.0.0.1'
PORT = 6000
clients = []

# --- Bagian komputasi paralel ---
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

def parallel_computation():
    data = None
    if rank == 0:
        arr = np.arange(10, dtype='i')
        chunks = np.array_split(arr, size)
        data = chunks
    else:
        arr = None

    # Scatter: bagi data ke semua proses
    local_data = comm.scatter(data, root=0)
    local_sum = np.sum(local_data)

    # Gather hasil penjumlahan
    total_sum = comm.reduce(local_sum, op=MPI.SUM, root=0)

    if rank == 0:
        return total_sum
    return None

def handle_client(conn, addr):
    print(f"[TERHUBUNG] Client dari {addr}")
    clients.append(conn)
```

```python
import socket

HOST = '127.0.0.1'
PORT = 6000

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((HOST, PORT))

print("Terhubung ke server. Menunggu broadcast hasil komputasi...")

while True:
    data = client.recv(1024).decode('utf-8')
    if not data:
        break
    print(f"ini adalah pesan dari server {data}")
```

Berikut potongan code di sisi client

```python
def broadcast(message):
    for c in clients:
        try:
            c.sendall(message.encode('utf-8'))
        except:
            clients.remove(c)
            c.close()

def main():
    if rank == 0:
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.bind((HOST, PORT))
        server.listen()
        print(f"[SERVER] Menunggu client di {HOST}:{PORT}")

        # Thread untuk menerima client
        threading.Thread(target=lambda: accept_clients(server), daemon=True).start()

        while True:
            input("Tekan ENTER untuk jalankan komputasi paralel...")
            result = parallel_computation()
            if result is not None:
                msg = f"Hasil komputasi paralel: {result}"
                print(f"[BROADCAST] {msg}")
                broadcast(msg)

def accept_clients(server):
    while True:
        conn, addr = server.accept()
        threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()

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