Penjelasan

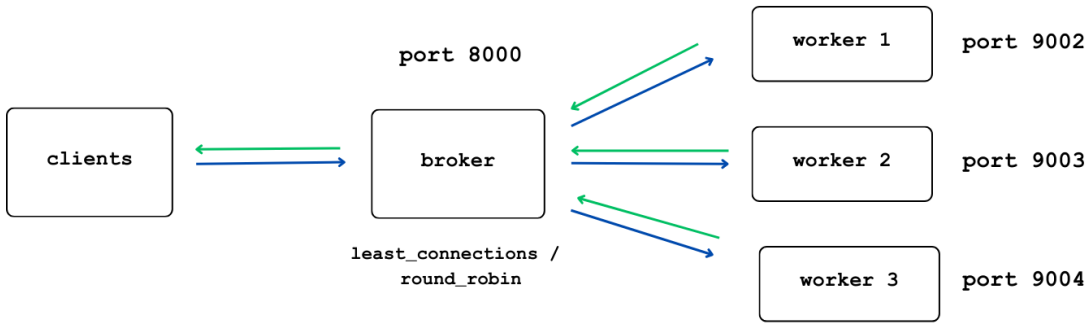
Definisi :

Load balancing adalah teknik untuk mendistribusikan beban kerja atau lalu lintas jaringan secara merata di beberapa server atau sumber daya komputasi. Load balancer bertindak sebagai "penjaga pintu" yang menentukan ke server atau worker mana permintaan akan dikirim, dengan tujuan menjaga performa sistem, meminimalkan waktu respons, dan meningkatkan ketersediaan layanan.

Cara Kerja Load Balancing

Load balancer bertindak sebagai perantara antara pengguna (client) dan server yang mengelola permintaan. Saat permintaan masuk dari client, load balancer memilih worker atau server berdasarkan algoritma tertentu (seperti round-robin, least connections, atau weighted distribution) untuk menangani permintaan tersebut. Situs web besar menggunakan load balancing agar dapat mengarahkan pengguna ke server yang paling optimal, sehingga waktu muat halaman tetap cepat meskipun banyak pengguna mengakses situs secara bersamaan.

Flowchart



Komponen:

- Client: Mengirimkan permintaan ke sistem, bisa berupa pengguna yang mengakses aplikasi web, atau layanan lain yang membuat permintaan ke sistem. Dalam arsitektur ini, client tidak langsung berhubungan dengan worker, tetapi melalui load balancer atau broker.
- Broker: Broker bertindak sebagai perantara atau "penghubung" antara client dan worker, membantu mendistribusikan tugas dalam menerima permintaan dari client dan menentukan worker mana yang akan menangani permintaan tersebut.
- Worker: Menangani tugas atau permintaan dan mengembalikan hasil ke broker untuk diteruskan ke client.

Broker menggunakan dua metode untuk memilih worker:

- Least Connections: Mengarahkan permintaan ke worker dengan jumlah koneksi terkecil (beban paling ringan). Mengarahkan permintaan ke server dengan jumlah koneksi aktif paling sedikit, cocok untuk aplikasi dengan sesi yang panjang.
- Round Robin: Mengirim permintaan secara bergilir ke setiap worker. Setiap permintaan secara berurutan diarahkan ke setiap server tanpa mempertimbangkan status atau beban dari server.

Penjelasan Alur Komunikasi

- Client mengirimkan permintaan, yang mencakup tipe aplikasi (Long atau Short) ke Broker.
- Broker memilih worker berdasarkan metode yang dipilih:
  Jika least_connections, broker memeriksa jumlah koneksi setiap worker dan memilih yang memiliki koneksi paling sedikit.
  Jika round_robin, broker bergilir memilih setiap worker secara bergantian.
- Broker mengirim permintaan ke Worker yang dipilih.
- Worker memproses permintaan:
  Long akan mengambil waktu 5 detik, sedangkan Short akan mengambil 1 detik.
- Setelah Worker selesai, ia mengirimkan respons kembali ke Broker.
- Broker meneruskan respons dari Worker ke Client.

Flow client.py

1. Membaca application_id dan approach dari command line.
2. Membuka client socket.
3. Menghubungkan ke broker di localhost pada port 8000
4. Klien mengirimkan data permintaan ke broker dalam format AppID:{application_id}:{approach}.
5. Broker kemudian akan meneruskan permintaan ini ke salah satu worker berdasarkan metode balancing yang dipilih (least_connections atau round_robin).
6. Setelah worker selesai memproses permintaan, ia mengirim respons ke broker, yang kemudian diteruskan ke klien.
7. Klien menerima dan menampilkan respons dari worker, lalu koneksi ditutup.

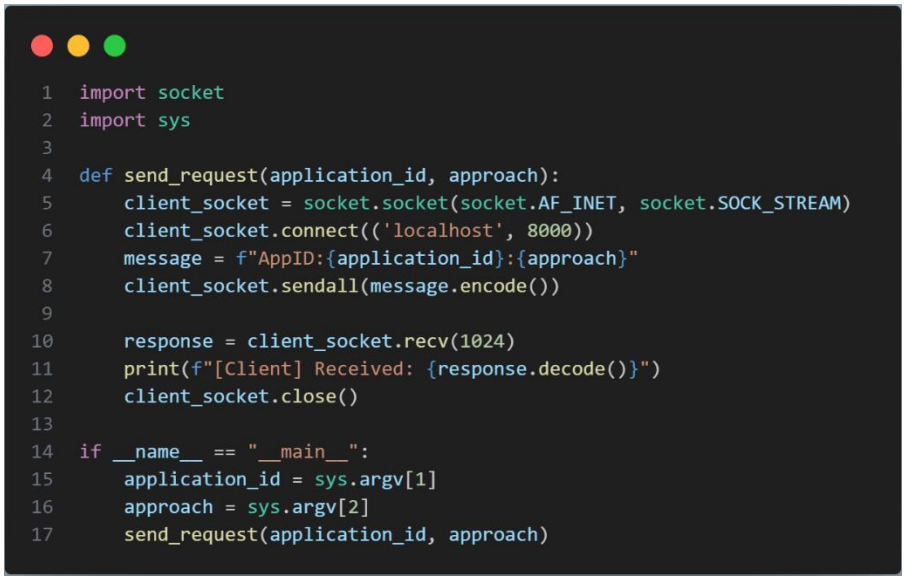
Flow broker.py

1. Menginisialisasi alamat dan koneksi worker.
2. Mendefinisikan fungsi pemilihan worker berdasarkan least_connections atau round_robin.
3. Broker mendengarkan permintaan klien pada port 8000.
4. Saat klien terhubung, broker menerima data permintaan yang berisi tipe aplikasi dan pendekatan load balancing.
5. Berdasarkan pendekatan balancing, broker memilih worker menggunakan least_connections atau round_robin.
6. Broker meneruskan permintaan ke worker yang dipilih dan menunggu respons.
7. Setelah menerima respons dari worker, broker mengirimkan respons ini kembali ke klien.
8. Jika least_connections digunakan, broker memperbarui jumlah koneksi pada worker yang dipilih untuk melacak beban masing-masing worker.

Flow worker.py

1. Worker mendengarkan koneksi pada port tertentu yang ditentukan saat skrip dijalankan.
2. Saat menerima koneksi dari broker, worker memeriksa tipe permintaan (Long atau Short).
3. Setelah pemrosesan selesai, worker mengirimkan respons ke broker dan menutup koneksi.
4. Menggunakan threading untuk menangani permintaan secara paralel.

Penjelasan file client.py



```python
import socket
import sys

def send_request(application_id, approach):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect(('localhost', 8000))
    message = f"AppID:{application_id}:{approach}"
    client_socket.sendall(message.encode())

    response = client_socket.recv(1024)
    print(f"[Client] Received: {response.decode()}")
    client_socket.close()

if __name__ == "__main__":
    application_id = sys.argv[1]
    approach = sys.argv[2]
    send_request(application_id, approach)
```

- Socket : Digunakan untuk mengatur komunikasi jaringan (TCP/IP).
- Sys : Digunakan untuk mengambil argumen dari baris perintah.
- send_request(application_id, approach) : Fungsi ini bertanggung jawab untuk mengirim permintaan dari klien ke broker dan menerima respons.
- client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  Membuat objek soket client_socket dengan keluarga alamat AF_INET (IPv4) dan tipe soket SOCK_STREAM (TCP).
- client_socket.connect(('localhost', 8000))
  Menghubungkan soket klien ke server broker yang berjalan di localhost pada port 8000.
- message = f"AppID:{application_id}:{approach}"
  Membuat pesan yang akan dikirim ke broker, berisi AppID (misalnya, "Long" atau "Short") dan metode balancing (approach).
- client_socket.sendall(message.encode())
  Mengirim pesan ke broker. Fungsi sendall() memastikan seluruh pesan dikirim. Pesan dienkode ke byte menggunakan encode().
- response = client_socket.recv(1024)
  Menerima respons dari broker, dengan batas maksimal ukuran data yang diterima sebesar 1024 byte.
- print(f"[Client] Received: {response.decode()}")
  Menampilkan respons yang diterima dari broker di terminal setelah diubah kembali ke format string dengan decode().

- client_socket.close()
  Menutup koneksi soket setelah respons diterima.
- application_id = sys.argv[1]
  approach = sys.argv[2]
  send_request(application_id, approach)
  Menangkap argumen dari baris perintah Argumen (sys.argv[1]) adalah application_id yang menentukan tipe aplikasi (Long atau Short), dan
  argumen (sys.argv[2]) adalah approach yang menentukan pendekatan load balancing (least_connections atau round_robin).
- send_request() kemudian dipanggil dengan application_id dan approach sebagai parameter.

Penjelasan file broker.py

```python
import socket
import threading

# Worker Server Configurations
WORKER_ADDRESSES = [("localhost", 9002), ("localhost", 9003), ("localhost", 9004)]
worker_connections = [0, 0, 0]  # Track the load of each worker

def select_worker(approach):
    if approach == "least_connections":
        # Find the worker with the least connections
        return worker_connections.index(min(worker_connections))
    elif approach == "round_robin":
        # Find the worker based on round-robin approach
        select_worker.counter = (select_worker.counter + 1) % len(WORKER_ADDRESSES)
        return select_worker.counter

select_worker.counter = -1

def broker_server():
    broker_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    broker_socket.bind(('localhost', 8000))
    broker_socket.listen()
    print("[Broker] Listening on port 8000...")

    while True:
        client_conn, client_addr = broker_socket.accept()
        print(f"[Broker] Connected with {client_addr}")
        data = client_conn.recv(1024).decode()
        app_type = data.split(":")[1].strip()

        approach = data.split(":")[2].strip()
        worker_id = select_worker(approach)
        worker_host, worker_port = WORKER_ADDRESSES[worker_id]

        try:
            worker_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            worker_conn.connect((worker_host, worker_port))
            worker_conn.sendall(f"AppID:{app_type}".encode())

            response = worker_conn.recv(1024)
            print(f"[Broker] Response from Worker-{worker_id + 1}: {response.decode()}")
            client_conn.sendall(response)

            # Update connection counter based on approach
            worker_connections[worker_id] += 1 if approach == "least_connections" else 0

        except ConnectionError:
            print(f"[Broker] Failed to connect to Worker-{worker_id + 1}")

        client_conn.close()
        worker_conn.close()

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

- threading : Dapat digunakan jika ingin mengelola beberapa koneksi secara paralel. Namun, saat ini tidak diimplementasikan dalam kode ini.
- WORKER_ADDRESSES = [("localhost", 9002), ("localhost", 9003), ("localhost", 9004)] Daftar alamat dan port worker yang tersedia.
- worker_connections: Daftar yang melacak jumlah koneksi saat ini untuk setiap worker, digunakan dalam metode least_connections.
- select_worker(approach) : Fungsi ini memilih worker berdasarkan metode balancing yang dipilih (least_connections atau round_robin).
- least_connections: Mencari worker dengan jumlah koneksi aktif paling sedikit.

- round_robin: Menggunakan pendekatan round-robin (bergiliran) untuk mendistribusikan permintaan, dengan menjaga indeks counter yang berputar di antara worker.
- select_worker.counter = -1
  Inisialisasi counter untuk metode round-robin.
- broker_server() : Fungsi utama yang menangani koneksi dari klien dan meneruskannya ke worker.
- broker_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  broker_socket.bind(('localhost', 8000))
  broker_socket.listen()
  print("[Broker] Listening on port 8000...")
  Membuat soket broker yang mendengarkan pada localhost di port 8000.
- client_conn, client_addr = broker_socket.accept()
  Menerima koneksi dari klien dan menampilkan alamat klien yang terhubung.
- data = client_conn.recv(1024).decode()
  app_type = data.split(":")[1].strip()
  approach = data.split(":")[2].strip()
  Menerima data dari klien, menguraikannya untuk mendapatkan app_type dan approach yang akan digunakan dalam load balancing.
- worker_id = select_worker(approach)
  worker_host, worker_port = WORKER_ADDRESSES[worker_id]
  Memanggil fungsi select_worker() untuk mendapatkan ID worker yang dipilih sesuai dengan metode balancing (least_connections atau round_robin).
  Mendapatkan alamat dan port worker yang dipilih.
- worker_conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  worker_conn.connect((worker_host, worker_port))
  worker_conn.sendall(f"AppID:{app_type}".encode())
  Membuat koneksi ke worker yang dipilih dan meneruskan permintaan dari klien.
- response = worker_conn.recv(1024)
  print(f"[Broker] Response from Worker-{worker_id + 1}: {response.decode()}")
  client_conn.sendall(response)
  Menerima respons dari worker, menampilkannya, dan kemudian mengirimkan kembali respons ini ke klien.
  Memperbarui Jumlah Koneksi Worker (hanya untuk least_connections):
- worker_connections[worker_id] += 1 if approach == "least_connections" else 0
  Jika pendekatan yang digunakan adalah least_connections, maka jumlah koneksi pada worker yang dipilih diperbarui untuk mencatat beban koneksi saat ini.
- except ConnectionError:
  print(f"[Broker] Failed to connect to Worker-{worker_id + 1}"
  client_conn.close()
  worker_conn.close()
  Jika terjadi kegagalan koneksi ke worker, broker menampilkan pesan kesalahan. Koneksi klien dan koneksi worker ditutup di akhir.

Penjelasan worker.py

```python
import socket
import time
import threading

def handle_request(connection, address, app_type):
    if app_type == "Long":
        print(f"[Worker] Processing long request from {address}")
        time.sleep(5)  # Simulate a long computation
    elif app_type == "Short":
        print(f"[Worker] Processing short request from {address}")
        time.sleep(1)  # Simulate a short computation
    connection.sendall(b"Response from Worker")
    connection.close()

def worker_server(port, worker_id):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(('localhost', port))
    server_socket.listen()
    print(f"[Worker-{worker_id}] Listening on port {port}...")

    while True:
        conn, addr = server_socket.accept()
        data = conn.recv(1024).decode()
        app_type = data.split(":")[1].strip()
        threading.Thread(target=handle_request, args=(conn, addr, app_type)).start()

if __name__ == "__main__":
    import sys
    worker_id = sys.argv[1]
    worker_port = 9001 + int(worker_id)
    worker_server(worker_port, worker_id)
```

- time: Untuk mensimulasikan waktu pemrosesan permintaan yang berbeda (panjang atau pendek) menggunakan sleep.
- threading: Untuk menangani setiap permintaan dalam thread terpisah, memungkinkan worker menangani beberapa permintaan secara bersamaan.
- handle_request(connection, address, app_type): Fungsi ini menangani setiap permintaan klien, mensimulasikan pemrosesan panjang atau pendek berdasarkan jenis permintaan (Long atau Short), dan kemudian mengirim respons.
- app_type adalah tipe permintaan yang diterima (Long atau Short):
- Setelah pemrosesan selesai, worker mengirimkan respons ke klien berupa "Response from Worker" dan menutup koneksi.
- Fungsi worker_server(port, worker_id): Fungsi ini memulai server worker yang mendengarkan permintaan dari broker.
- server_socket: Soket utama yang digunakan untuk mendengarkan koneksi.
- bind(('localhost', port)): Mengikat soket ke localhost dengan port tertentu (ditentukan saat menjalankan skrip).
- listen(): Mengatur soket untuk mulai mendengarkan koneksi masuk.
- Loop utama while True menunggu koneksi baru dari broker.
- conn, addr = server_socket.accept(): Menerima koneksi dari broker, dengan conn sebagai objek koneksi dan addr sebagai alamat klien.

- data = conn.recv(1024).decode(): Menerima data dari broker dan mendekodenya menjadi teks. Data ini berisi tipe aplikasi (Long atau Short) yang akan diproses.
- app_type = data.split(":")[1].strip(): Menguraikan data untuk mendapatkan jenis permintaan.
- threading.Thread(target=handle_request, args=(conn, addr, app_type)).start(): Membuat thread baru untuk menangani setiap permintaan, memungkinkan worker menangani beberapa permintaan secara bersamaan.

Simulasi

1. Pastikan di laptop sudah terinstall phyton
2. Jalankan semua worker di terminal yang berbeda dengan perintah
   python worker.py 1
   python worker.py 2
   python worker.py 3

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\xampp\htdocs\Tugas 1> cd codes
PS C:\xampp\htdocs\Tugas 1\codes> python worker.py 1
[Worker-1] Listening on port 9002...
[Worker] Processing short request from ('127.0.0.1', 57731)
[Worker] Processing long request from ('127.0.0.1', 58051)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\xampp\htdocs\Tugas 1> cd codes
PS C:\xampp\htdocs\Tugas 1\codes> python worker.py 2
[Worker-2] Listening on port 9003...
[Worker] Processing long request from ('127.0.0.1', 58055)
[Worker] Processing short request from ('127.0.0.1', 58058)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\xampp\htdocs\Tugas 1> cd codes
PS C:\xampp\htdocs\Tugas 1\codes> python worker.py 3
[Worker-3] Listening on port 9004...
[Worker] Processing short request from ('127.0.0.1', 58066)
```

3. Jalankan broker, pastikan setiap worker berada dalam kondisi "listening" pada port yang telah ditentukan
   python broker.py

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\xampp\htdocs\Tugas 1\codes> python broker.py
[Broker] Listening on port 8000...
[Broker] Connected with ('127.0.0.1', 57730)
[Broker] Response from Worker-1: Response from Worker
[Broker] Connected with ('127.0.0.1', 58050)
[Broker] Response from Worker-1: Response from Worker
[Broker] Connected with ('127.0.0.1', 58054)
[Broker] Response from Worker-2: Response from Worker
[Broker] Connected with ('127.0.0.1', 58057)
[Broker] Response from Worker-2: Response from Worker
[Broker] Connected with ('127.0.0.1', 58065)
[Broker] Response from Worker-3: Response from Worker
```

4. Jalankan client dengan perintah
   python client.py Long least_connections
   python client.py Short round_robin

```
PS C:\xampp\htdocs\Tugas 1\codes> python client.py Short round_robin
[Client] Received: Response from Worker
PS C:\xampp\htdocs\Tugas 1\codes> python client.py Long least_connections
[Client] Received: Response from Worker
PS C:\xampp\htdocs\Tugas 1\codes> python client.py Long round_robin
[Client] Received: Response from Worker
PS C:\xampp\htdocs\Tugas 1\codes> python client.py Short least_connections
[Client] Received: Response from Worker
PS C:\xampp\htdocs\Tugas 1\codes> python client.py Short round_robin
[Client] Received: Response from Worker
PS C:\xampp\htdocs\Tugas 1\codes>
```

   Perintah dapat diganti Long atau Short serta least_connections atau round_robin untuk mencoba berbagai kombinasi permintaan dari client
   - Jika Long, worker mensimulasikan pemrosesan panjang dengan time.sleep(5).
   - Jika Short, worker mensimulasikan pemrosesan pendek dengan time.sleep(1).

   Jika menggunakan
   - Round Robin: Setiap permintaan secara berurutan diarahkan ke setiap server tanpa mempertimbangkan status atau beban dari server.
   - Least Connections: Mengarahkan permintaan ke server dengan jumlah koneksi aktif paling sedikit, cocok untuk aplikasi dengan sesi yang panjang.

   Gambar di atas menunjukkan hasil dari beberapa kombinasi permintaan yang dilakukan selama simulasi ini.