Nama: Nur Iliyanie

NIM: 3312501045

Kelas: IF 1B – Pagi

Soal

1. Sebuah sistem cloud memiliki 5 server, 20 file, dan setiap file disimpan di 2 server untuk redundansi.
   a. Tentukan banyak kombinasi penyimpanan yang mungkin.
   b. Analisis kemungkinan tabrakan data (collision).
   c. Buat simulasi logika / pseudocode deteksi duplikasi.
2. Apa keterkaitan antara konsep kombinasi dan desain sistem penyimpanan data?
3. Mengapa prinsip sarang merpati menjadi penting dalam pemrograman dan struktur data?

1. Diketahui

- Jumlah Server (N): 5
- Jumlah File (F): 20
- Redundansi: Setiap file disimpan di 2 server (k): 2

Ditanya

a) Banyak kombinasi penyimpanan yang mungkin.
b) Analisis kemungkinan tabrakan data (*collision*).
c) Simulasi logika/pseudocode deteksi duplikasi.

Jawaban

a. Banyak Kombinasi Penyimpanan yang Mungkin

1. Kombinasi Pasangan Server untuk Satu File:

Banyaknya cara memilih 2 server dari 5 yang tersedia dihitung dengan rumus kombinasi C(N, k):

$$C(5,2) = \frac{5!}{2!(5-2)!} = \frac{5 \times 4}{2 \times 1} = 10$$

Ada 10 kemungkinan pasangan server.

2. Total Kombinasi Penyimpanan untuk 20 File:
   Karena penyimpanan setiap file independen dan ada 20 file, total kombinasinya adalah:

$$\text{Total Kombinasi} = (C(5,2))^F = 10^{20}$$

Banyak kombinasi penyimpanan yang mungkin adalah 10^{20} cara.

b. Analisis Kemungkinan Tabrakan Data (Collision)

Kemungkinan tabrakan data (dua file menggunakan pasangan server yang sama) adalah pasti (certain) dan sangat tinggi.

- Ini didasarkan pada Prinsip Sarang Merpati (Pigeonhole Principle).

  Merpati (N) : 20 File

  Sarang (M) : 10 Pasangan Server Unik

- Karena jumlah file (20) lebih besar dari jumlah pasangan server unik (10), setidaknya satu pasangan server pasti akan menyimpan dua file atau lebih.
- Implikasi Desain: Tabrakan ini menyebabkan ketidakseimbangan beban kerja antar server dan mengurangi efektivitas redundansi karena kegagalan satu pasangan server dapat menghilangkan akses ke banyak file.

c. Simulasi Logika/Pseudocode Deteksi Duplikasi

Logika ini digunakan untuk mendeteksi duplikasi file di sistem cloud. Setiap file dicek berdasarkan hash kontennya. Jika konten sudah ada, sistem memeriksa apakah server

tempat penyimpanan sama sudah digunakan; jika ya, penyimpanan diblok untuk menghindari duplikasi lokasi, dan jika tidak, lokasi server baru ditambahkan untuk redundansi. File unik akan disimpan dan dicatat sebagai entri baru dalam indeks global.

```
# Struktur Data Global

# GLOBAL_INDEX: Dictionary { file_hash : list_of_server_pairs }
GLOBAL_INDEX = {}

def simpan_file_baru(file_data, server_pair):
    """
    file_data   : konten file
    server_pair : tuple(server1, server2)
    """

    # 1. Hitung hash file (untuk deteksi duplikat konten)
    file_hash = hitung_hash(file_data)  # fungsi hash (misal SHA256)

    # 2. Periksa apakah file sudah ada berdasarkan hash
    if file_hash in GLOBAL_INDEX:
        # File dengan konten sama sudah ada
        if server_pair in GLOBAL_INDEX[file_hash]:
            # Lokasi server sama → duplikat lokasi
            print(f"DUPLIKASI LOKASI: file sudah ada di {server_pair}")
            return "GAGAL: DUPLIKASI_LOKASI"
        else:
            # Konten sama tapi server baru → tambahkan lokasi baru
            GLOBAL_INDEX[file_hash].append(server_pair)
            simpan_file(file_data, server_pair)  # simpan fisik di server
baru
            print(f"FILE DUPLIKAT, lokasi baru ditambahkan:
{server_pair}")
            return "SUKSES: LOKASI BARU DITAMBAHKAN"
    else:
        # File unik → simpan dan buat entri baru di indeks
        GLOBAL_INDEX[file_hash] = [server_pair]
        simpan_file(file_data, server_pair)
        print(f"FILE UNIK BERHASIL DISIMPAN di {server_pair}")
        return "SUKSES: FILE_UNIK_DISIMPAN"

# Fungsi tambahan contoh (implementasi fisik tergantung sistem)
def hitung_hash(file_data):
    import hashlib
    return hashlib.sha256(file_data.encode()).hexdigest()
```

```
def simpan_file(file_data, server_pair):
    # Simulasi penyimpanan fisik
    print(f"Menyimpan file di server {server_pair}")
```

2. Keterkaitan antara kombinasi dan desain sistem penyimpanan data (terutama untuk redundansi/replikasi) adalah fundamental dan dapat dilihat dari aspek-aspek berikut:
    1. **Redundansi dan Toleransi Kegagalan (Fault Tolerance):**
        - **Konsep Kombinasi:** Kombinasi menentukan **berapa banyak cara** replika data dapat didistribusikan ke sejumlah node (server). Jika sistem memiliki N server dan membutuhkan k replika per file, C(N, k) adalah jumlah total *pasangan/grup* server yang mungkin untuk menyimpan data.
        - **Desain Sistem:** Jumlah kombinasi ini membantu desainer sistem untuk memahami jumlah maksimum set data unik yang dapat dipertahankan. Semakin banyak kombinasi yang mungkin, semakin fleksibel dan beragam distribusi replika, yang dapat mengurangi korelasi kegagalan (yaitu, mengurangi kemungkinan dua file penting berbagi replika yang sama).
    2. **Pemetaan Data dan Distribusi Beban (Data Mapping and Load Balancing):**
        - **Konsep Kombinasi:** Kombinasi yang dihasilkan oleh C(N, k) merepresentasikan **"**alamat" unik di mana data dapat disimpan.
        - **Desain Sistem:** Desainer menggunakan konsep ini (sering kali dengan hashing konsisten atau algoritma *placement*) untuk memetakan setiap file ke salah satu kombinasi server yang tersedia. Tujuannya adalah memilih algoritma pemetaan yang akan mendistribusikan file secara merata ke seluruh C(N, k) kombinasi yang ada. Hal ini sangat penting untuk menyeimbangkan beban kerja (load balancing) di semua server secara efisien.
    3. **Optimalisasi Biaya:**
        - **Konsep Kombinasi:** Memilih nilai k (derajat redundansi) adalah trade-off. Semakin besar k, semakin banyak ruang penyimpanan yang terbuang (biaya), tetapi semakin besar ketersediaan data.

- **Desain Sistem:** Analisis kombinasi membantu sistem engineer untuk menentukan nilai k optimal yang memenuhi persyaratan ketersediaan data minimum sambil meminimalkan biaya penyimpanan.

3. Prinsip Sarang Merpati (PSM) sangat penting karena ia adalah alat teoritis fundamental untuk memprediksi konflik dan membuktikan batasan performa sistem komputasi.

   1. Prediksi Tabrakan (Collision) dalam Hashing

   Pentingnya: PSM secara formal menjelaskan bahwa jika memiliki lebih banyak *keys* (merpati, data) daripada *slot* yang tersedia di *hash table* (sarang), tabrakan (collision) pasti terjadi.

   Implikasi: Prinsip ini memaksa pengembang untuk merancang dan mengimplementasikan mekanisme penanganan tabrakan yang canggih (*chaining* atau *open addressing*) untuk menjaga kinerja *hash table* tetap optimal.

   2. **Membuktikan Batas Bawah Algoritma**

   Pentingnya: Dalam algoritma Sorting Berbasis Perbandingan (seperti Merge Sort), PSM membantu membuktikan bahwa kompleksitas waktu terburuknya (worst-case time complexity) adalah $\Omega(N \log N)$.

   Implikasi: Ini menetapkan batas teoritis (batas bawah) yang tidak dapat dilampaui oleh algoritma apa pun yang bekerja dengan perbandingan.

   3. Batasan Kompresi Data

   Pentingnya: PSM menjelaskan mengapa kompresi *lossless* yang *selalu* menghasilkan output yang lebih kecil tidak mungkin. Jika ada lebih banyak *input* yang mungkin (N) daripada *output* yang lebih kecil yang mungkin (M), maka pasti ada beberapa input berbeda yang menghasilkan output yang sama, yang berarti tidak dapat dikembalikan ke keadaan semula.

   Prinsip ini adalah dasar untuk memahami batasan fundamental dalam desain struktur data dan efisiensi algoritma.