**NAME:** *Jaslin jeba thurai j*

**REG NO:** *192321175*

**Assignment: Resource Optimization in Data Centres**

**Scenario**

A data centre needs to optimize power consumption and server usage to reduce operational costs. Efficient resource allocation is essential for balancing workloads across servers without overloading any single server. The company seeks an algorithmic solution that can handle high data volumes while minimizing power usage and operational expenses.

**Tasks**

1. Analyse the Time Complexity of a Brute-Force Approach to Resource Allocation

Problem Statement: Given a set of workloads and a set of servers, the goal is to allocate workloads to servers such that the total power consumption is minimized while ensuring that no server exceeds its capacity.

Brute-Force Approach: The brute-force method involves generating all possible allocations of workloads to servers and calculating the total power consumption for each allocation.

**Time Complexity:**

Let ( n ) be the number of workloads and ( m ) be the number of servers.

The total number of possible allocations is ( m^n ) (each workload can go to any of the ( m ) servers).

Therefore, the time complexity of the brute-force approach is ( O(m^n) ), which is exponential and impractical for large ( n ).

1. Prove the Correctness of Load Balancing Algorithms and Analyse Their Runtime

**Greedy Algorithm:** A common greedy approach is to assign each workload to the server with the least current load.

Correctness: This method is correct in the sense that it aims to keep all servers balanced, but it may not always yield the optimal solution.

**Runtime Analysis:** The greedy algorithm runs in ( O(n \log m) ) time, where ( n ) is the number of workloads and ( m ) is the number of servers, due to the need to find the server with the least load.

1. Implement Dynamic Programming and Greedy Solutions to Distribute Workloads Across Servers

**Dynamic Programming Approach**: Implement a solution that uses dynamic programming to find the optimal allocation of workloads to servers.

**Greedy Approach:** Implement the greedy algorithm as described above.

1. Use Backtracking to Adjust Resource Allocation in Response to Server Constraints

Implement a backtracking algorithm that can adjust resource allocation dynamically based on server constraints (e.g., maximum load, power limits).

1. Assess the Effectiveness of Polynomial and Non-Polynomial Algorithms for Scalability

Compare the performance of polynomial-time algorithms (like dynamic programming and greedy) against non-polynomial algorithms (like brute-force) in terms of scalability and efficiency.

**Deliverables**

1. **Code Implementations**

Provide code implementations for the following: Brute-force load balancing algorithm Dynamic programming load balancing algorithm Greedy load balancing algorithm Backtracking algorithm for resource allocation adjustment.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#define MAX\_SERVERS 10

#define MAX\_WORKLOADS 20

Void bruteForce(int workloads[], int n, int m);

Void greedyLoadBalancing(int workloads[], int n, int m);

Int dynamicProgramming(int workloads[], int n, int m, int serverCapacity[]);

Void bruteForce(int workloads[], int n, int m) {

Int totalAllocations = 1;

For (int I = 0; I < n; i++) {

totalAllocations \*= m; // m^n

}

Printf(“Total possible allocations (Brute-force): %d\n”, totalAllocations);

}

Void greedyLoadBalancing(int workloads[], int n, int m) {

Int serverLoad[MAX\_SERVERS] = {0};

For (int I = 0; I < n; i++) {

Int minLoadIndex = 0;

For (int j = 1; j < m; j++) {

If (serverLoad[j] < serverLoad[minLoadIndex]) {

minLoadIndex = j;

}

}

serverLoad[minLoadIndex] += workloads[i];

printf(“Workload %d assigned to Server %d. Current load: %d\n”, workloads[i], minLoadIndex, serverLoad[minLoadIndex]);

}

}

Int dynamicProgramming(int workloads[], int n, int m, int serverCapacity[]) {

Int.dp[MAX\_SERVERS][MAX\_WORKLOADS] = {0};

For (int I = 0; I <= m; i++) {

For (int j = 0; j <= n; j++) {

Dp[i][j] = INT\_MAX;

}

}

Dp[0][0] = 0;

For (int I = 1; I <= m; i++) {

For (int j = 0; j < n; j++) {

For (int k = 0; k <= j; k++) {

If (dp[i-1][k] + workloads[j] <= serverCapacity[i-1]) {

Dp[i][j+1] = dp[i-1][k] + workloads[j];

}

}

}

}

Return dp[m][n];

}

Int main() {

Int workloads[] = {10, 20, 30, 40, 50};

Int n = sizeof(workloads) / sizeof(workloads[0]);

Int m = 3;

Int serverCapacity[] = {100, 100, 100};

Printf(“Brute-force approach:\n”);

bruteForce(workloads, n, m);

printf(“\nGreedy load balancing approach:\n”);

greedyLoadBalancing(workloads, n, m);

printf(“\nDynamic programming approach:\n”);

int minLoad = dynamicProgramming(workloads, n, m, serverCapacity);

printf(“Minimum load on last server: %d\n”, minLoad);

return 0;

}

1. **Comparative Report**

Write a report comparing the performance, scalability, and resource efficiency of the different algorithms implemented.

**Include**: Time complexity analysis Space complexity analysis Practical considerations for real-world applications.

1. **Performance Graphs**

**Generate performance graphs showing:** Power usage across different data volumes. Load distribution across servers for each algorithm.Include a discussion of the results and any observed trends.

**Submission Guidelines**: Submit the code implementations in a single zip file. Include the comparative report in PDF format. Ensure that all graphs are clearly labeled and included in the report.

**Deadline for submission**: [Insert Deadline Here]

This assignment format provides a structured approach to tackling the problem of resource optimization in data centres, ensuring that all aspects of the task are covered comprehensively.