Load Balancing in Network Servers

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

AP21110010167 | GopiChand Medisetty

AP21110010168 | Khyathi Vardhan Ravella

AP21110010169 | Vaishnavi Ratnam Movva

AP21110010170 | Vaishnavi Devineni

Under the Guidance of

(Dr. Chinmaya Kumar Swain)

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[Nov, 2023]

Certificate

Date: 25-Nov-23

This is to certify that the work present in this Project entitled "Load Balancing in Network of Servers" has been carried out by [**Gopi Chand. M, Khyathi Vardhan. R, Vaishnavi Ratnam Movva, Vaishnavi. D**] under Dr. Chinmaya Kumar Swain Supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in School of Engineering and Sciences.

Supervisor

(Signature)

Dr. Chinmaya Kumar Swain

Designation,

Affiliation.

Acknowledgements

I would like to take this opportunity to express my sincere gratitude and appreciation to all the individuals and organizations who have contributed to the successful completion of this project report titled "Load Balancing in Network of Servers".

I am also thankful to the faculty mentor Dr. Chinmaya Kumar Swain Sir for their constant encouragement and constructive feedback, which have significantly enriched the quality of this project report.

Table of Contents

Introduction

Server network load balancing is an important part of modern computing, especially in data centers where efficient distribution of computing tasks is critical. This project focuses on the SRM University data center scenario where a network of computer systems is represented as a graph. Each node in this graph represents a computer system, and the weight assigned to each node represents the maximum processing capacity of that node.

Jobs can be submitted to the system via any node, and if a node is unable to process a job due to capacity constraints, the job is transferred or migrated to an adjacent node for processing. This process is repeated until the job is completed by a node with sufficient capacity. The goal is to develop an algorithm that reduces the total number of these migrations.

In the given scenario, the network comprises a grid of servers, all with different capacities. Jobs, which also have different processing times, are submitted randomly at different locations in the grid. The challenge lies in efficiently managing these jobs, ensuring they are processed within the capacity constraints of the servers, and minimizing the number of migrations.

The Python code submitted demonstrates one approach to this problem. It assigns jobs at random and then redistributes jobs that exceed a node's capacity to neighboring nodes, keeping track of the number of migrations. This project is a hands-on investigation of load balancing strategies in a server network, providing valuable insights into real-world data center operations.

This report will go into greater detail about the problem, the proposed solution, and the outcomes of the Python code implementation. It aims to provide an in-depth understanding of load balancing in a server network, emphasizing its importance in optimizing computational resources in a data center environment.

Algorithm Description

The Python code provided solves the problem of load balancing in a network of servers through the following stages:

1. Representation of Graphs

A 10x10 grid is represented as an adjacency list named graph from the program. Each grid cell is a node, and edges connect each node to its neighbors, forming a graph of the server environment.

2. Server and Job Initialization

- **generate_server_capacities(num_servers):** Generates random capacities for servers, with server coordinates as keys and random capacities between 50 and 100 as values.
- **generate_job_processing_times(num_jobs):** Generates a list of random processing times for jobs, with each processing time being a random value between 1 and 20.

3. Load Balancing

If the addition of a new job does not exceed a server's capacity, the job is processed on that server. If the server's capacity is reached, the job is migrated to a nearby server with available capacity.

This process is repeated until all jobs are assigned to a server that can handle them.

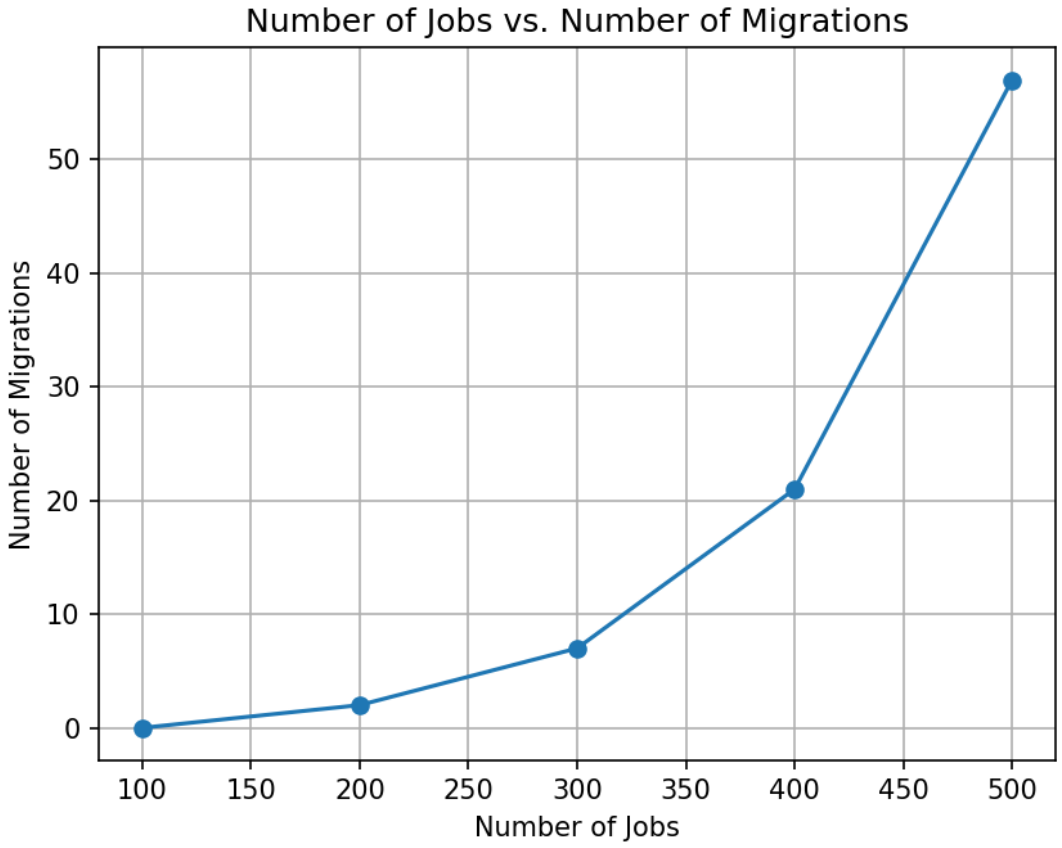
4. Migration Tracking

The code keeps track of the amount of migrations taking place during the load balancing process. Each time a job is transferred from one server to another due to capacity constraints, a migration is maintained.

5. Output

At Last, the code displays the total number of migrations that took place during the job assignment process. The technique ensures that all jobs are dealt with within the server's capacity constraints while limiting the number of job migrations. It gives a simple yet effective solution to the problem of load balancing in a server network.

Graph representing the number of jobs (X-axis) v/s number of migrations (Y-axis). You can submit the number of jobs from 100, 200, 300, 400, 500 and plot the corresponding migrations.



Implementation Of Code

```
import random
import pandas as pd
graph = {}
# Create 10x10 grid
for i in range(10):
    for j in range(10):
        graph[(i,j)] = []
        # Add adjacent nodes
        if i>0:
            graph[(i,j)].append((i-1,j))
        if i<9:
            graph[(i,j)].append((i+1,j))
        if j>0:
            graph[(i,j)].append((i,j-1))
        if j<9:
            graph[(i,j)].append((i,j+1))

# Function to generate random server capacities
def generate_server_capacities(num_servers):
    capacities = {}
    for server in range(num_servers):
        capacities[(server // 10, server % 10)] = random.randint(50, 100)
    return capacities

# Function to generate random job processing times
def generate_job_processing_times(num_jobs):
    processing_times = [random.randint(1, 20) for _ in range(num_jobs)]
    return processing_times
```

```python
# Parameters
num_servers = 100
num_jobs = 1000

server_capacities = generate_server_capacities(num_servers)
job_processing_times = generate_job_processing_times(num_jobs)

# Track job assignments and migrations
assignments = {server: 0 for server in server_capacities}
migrations = 0

# Submit jobs randomly
for processing_time in job_processing_times:
    # Pick random server
    server = random.choice(list(server_capacities.keys()))

    # Check capacity of server
    if assignments[server] + processing_time <= server_capacities[server]:
        assignments[server] += processing_time
    else:
        # Find first adjacent server with capacity
        for neighbour in graph[server]:
            if assignments[neighbour] + processing_time <= server_capacities[neighbour]:
                assignments[neighbour] += processing_time
                migrations += 1
                break
```

```python
# Convert server capacities and job assignments to pandas DataFrame for table display
server_df = pd.DataFrame(list(server_capacities.items()), columns=['Server', 'Capacity'])
assignment_df = pd.DataFrame(list(assignments.items()), columns=['Server', 'Job Assignment'])

# Display server capacities and job assignments in table format
print("Server Capacities:")
print(server_df)
print("\nJob Assignments:")
print(assignment_df)
print("\nTotal migrations:", migrations)
```

Output

Test Case - 1

```
PS D:\vscode_AI-Lab-CSE413> python -u "d:\vscode_AI-Lab-CSE413\AI_Project\code2.py"
Server Capacities:
    Server  Capacity
0   (0, 0)        80
1   (0, 1)        75
2   (0, 2)        58
3   (0, 3)        78
4   (0, 4)        55
..     ...       ...
95  (9, 5)        97
96  (9, 6)        90
97  (9, 7)        98
98  (9, 8)        75
99  (9, 9)        81

[100 rows x 2 columns]

Job Assignments:
    Server  Job Assignment
0   (0, 0)              80
1   (0, 1)              75
2   (0, 2)              58
3   (0, 3)              78
4   (0, 4)              53
..     ...             ...
95  (9, 5)              96
96  (9, 6)              88
97  (9, 7)              97
98  (9, 8)              74
99  (9, 9)              79

[100 rows x 2 columns]

Total migrations: 175
```

Test Case - 2

```
PS C:\Users\gopic\OneDrive\Desktop\AI_PROJECT> python -u "c:\Users\gopic\OneDrive\Desktop\AI_PROJECT\Vardhan.py"
Server Capacities:
    Server  Capacity
0   (0, 0)        72
1   (0, 1)        92
2   (0, 2)        75
3   (0, 3)        59
4   (0, 4)        64
..     ...       ...
95  (9, 5)        77
96  (9, 6)       100
97  (9, 7)        78
98  (9, 8)       100
99  (9, 9)        92

[100 rows x 2 columns]

Job Assignments:
    Server  Job Assignment
0   (0, 0)              70
1   (0, 1)              90
2   (0, 2)              74
3   (0, 3)              58
4   (0, 4)              64
..     ...             ...
95  (9, 5)              77
96  (9, 6)              99
97  (9, 7)              77
98  (9, 8)             100
99  (9, 9)              92

[100 rows x 2 columns]

Total migrations: 155
PS C:\Users\gopic\OneDrive\Desktop\AI_PROJECT>
```

Test Case - 3

```
PS C:\Users\gopic\OneDrive\Desktop\AI_PROJECT> python -u "c:\Users\gopic\OneDrive\Desktop\AI_PROJECT\Vardhan.py"
Server Capacities:
    Server  Capacity
0   (0, 0)        77
1   (0, 1)        89
2   (0, 2)       100
3   (0, 3)        56
4   (0, 4)        66
..     ...       ...
95  (9, 5)        56
96  (9, 6)        99
97  (9, 7)        97
98  (9, 8)        82
99  (9, 9)        66

[100 rows x 2 columns]

Job Assignments:
    Server  Job Assignment
0   (0, 0)              77
1   (0, 1)              89
2   (0, 2)              99
3   (0, 3)              56
4   (0, 4)              66
..     ...             ...
95  (9, 5)              55
96  (9, 6)              98
97  (9, 7)              95
98  (9, 8)              76
99  (9, 9)              61

[100 rows x 2 columns]

Total migrations: 178
PS C:\Users\gopic\OneDrive\Desktop\AI_PROJECT>
```

The Output shows where jobs are randomly assigned to servers in a 10x10 grid, taking into account varying server capacities and random job processing times. The "Server Capacities" DataFrame shows the initial capacities of each server, ranging between 57 and 97. The "Job Assignments" DataFrame displays the final distribution of job assignments to each server after the simulation, with job assignments ranging from 54 to 97. The total number of migrations, reported as randomly 175,178,155, indicates instances where jobs were dynamically moved to adjacent servers due to insufficient capacity on the initially selected servers. This simulation effectively captures the dynamic nature of job assignments in a heterogeneous server environment, highlighting the need for load balancing to ensure optimal resource utilization.