St. Francis Institute of Technology
Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII**
Subject: Distributed Computing **Class / Branch / Division: BE/CMPN/A**
Name: Rebecca Dias Roll Number:18

Experiment No: 07

AIM: Implementation of Load Balancing Algorithm.

Theory:

Load Balancing Algorithm:

Load balancing is the process of distributing the load among various nodes of a distributed system to improve both job response time and resource utilization while also avoiding a situation where some of the nodes are heavily loaded while other nodes are idle or lightly loaded..

Load-balancing approach: Type of load-balancing algorithms Static versus Dynamic: Static algorithms use only information about the average behavior of the system Static algorithms ignore the current state or load of the nodes in the system Dynamic algorithms collect state information and react to system state if it changed Static algorithms are much more simpler Dynamic algorithms are able to give significantly better performance

Load-balancing approach Type of static load-balancing algorithms Deterministic versus Probabilistic Deterministic algorithms use the information about the properties of the nodes and the characteristic of processes to be scheduled Probabilistic algorithms use information of static attributes of the system (e.g. number of nodes, processing capability, topology) to formulate simple process placement rules Deterministic approach is difficult to optimize Probabilistic approach has poor performance

Load-balancing approach Type of dynamic load-balancing algorithms Centralized versus Distributed Centralized approach collects information to server node and makes assignment decision Distributed approach contains entities to make decisions on a predefined set of nodes Centralized algorithms can make efficient decisions, have lower fault-tolerance Distributed algorithms avoid the bottleneck of collecting state information and react faster

Load-balancing approach Type of distributed load-balancing algorithms Cooperative versus Non-cooperative In Non-cooperative algorithms entities act as autonomous ones and make scheduling decisions independently from other entities In Cooperative algorithms distributed entities cooperate with each other Cooperative algorithms are more complex and involve larger overhead Stability of Cooperative algorithms are better .

Code:

```
def allocate(servers, processes): each =

    int(processes/servers) proc_array = [i+1

    for i in range(processes)] allocations = {}


    for i in range(servers):

        allocations[int(i)+1] = [] for

    i in range(len(allocations)):

        for j in range(each):

    allocations[int(i+1)].append("P"+str(proc_array.pop(0))) if(len(proc_array)

    != 0):

        try:

            for i in range(len(allocations)):

                allocations[int(i+1)].append("P"+str(proc_array.pop(0)))

        except Exception as e:

            pass

    return allocations


def print_allocations(allocations):

    for i in range(len(allocations)):

        print(f"Server:{i+1} -> {allocations[i+1]}")


servers = int(input("Enter the initial amount of servers >>>> ")) processes

= int(input("Enter the initial amount of processes >>>> "))


# Allocate
```

```
allocations = allocate(servers, processes) print("Initial
Allocation will be as shown below")
print_allocations(allocations)


ex_loop = ""
while (ex_loop.lower() != 's'):
    print("________________________CASES____________________________")
    print("1. Add servers 2. Add processes 3. Remove servers 4. Remove processes")
    print("................................................................................          ")
    case = int(input("Enter the case you want to execute >>>> "))
    if(case == 1):
        # Add New servers
        new_servers = int(input(
            f"Enter the new amount of servers to be added to old server count of {servers} >>>> "))
        servers = servers + new_servers allocations =
        allocate(servers, processes) print("New
        Allocation will be as shown below")
        print_allocations(allocations)
    elif(case == 2):
        # Add new processes
        new_procs = int(input(
            f"Enter the new amount of processes to be added to old process count of {processes} >>>
> "))
```

```
        processes = processes + new_procs allocations

        = allocate(servers, processes) print("New

        Allocation will be as shown below")

        print_allocations(allocations)

    elif(case == 3):

        # Remove servers new_servers = int(input( f"Enter the amount of servers to be removed

        from old server count of {servers} >>>> ")) servers = servers - new_servers allocations =

        allocate(servers, processes) print("New Allocation will be as shown below")

        print_allocations(allocations)

    elif(case == 4):

        # Remove processes

        new_procs = int(input(

            f"Enter the amount of processes to be removed from old process count of {processes} >>
>> "))

        processes = processes - new_procs allocations =

        allocate(servers, processes) print("New

        Allocation will be as shown below")

        print_allocations(allocations)


    ex_loop = input("To continue press 'c' else press 's' to stop >>>> ")
```

Output:

```
Enter the initial amount of servers >>>> 1

Enter the initial amount of processes >>>> 5
```

```
Initial Allocation will be as shown below

Server:1 -> ['P1', 'P2', 'P3', 'P4', 'P5']

_________________________CASES_____________________________
1. Add servers 2. Add processes 3. Remove servers 4. Remove processes
Enter the case you want to execute >>>> 1
Enter the new amount of servers to be added to old server count of 1 >>>> 1

New Allocation will be as shown below

Server:1 -> ['P1', 'P2', 'P5']
Server:2 -> ['P3', 'P4']
To continue press 'c' else press 's' to stop >>>> c

_________________________CASES_____________________________
…………………………………………………………………………………………
1. Add servers 2. Add processes 3. Remove servers 4. Remove processes
Enter the case you want to execute >>>> 4
Enter the amount of processes to be removed from old process count of 5 >>>> 1

New Allocation will be as shown below

Server:1 -> ['P1', 'P2']
Server:2 -> ['P3', 'P4']
```

Conclusion :

In this experiment, we learnt about the Load Balancing Approach In distributed computing. We came across different techniques such as Round Robin, Weighted Round Robin, Least Connection, Weighted Least Connection, Resource Based (Adaptive) etc and also implemented one.