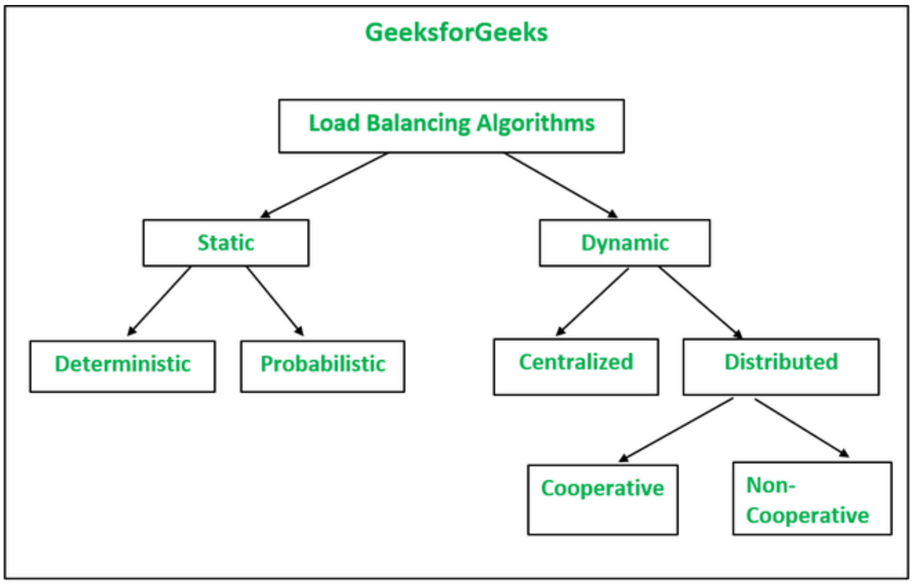
Distributed Computing

Experiment – 8

Program to demonstrate Load Balancing Algorithm in Java	
Learning Objective:	To study resource allocation and management.
Learning Outcome:	Ability to demonstrate Load Balancing Algorithm in Java.
Course Outcome:	**CSL801.5**
Program Outcome:	(**PO**1) **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. (**PO**2) **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. **(PO9) Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.** **(PO12) Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
Bloom's Taxonomy Level:	Analysis, Create
Theory:	**Load Balancing in Distributed Systems:** The Load Balancing approach refers to the division of load among the processing elements of a distributed system. The excess load of one processing element is distributed to other processing elements that have less load according to the defined limits. In other words, the load is maintained at each processing element in such a manner that neither it gets overloaded nor idle during the execution of a program to maximize the system throughput which is the ultimate goal of distributed systems. This approach makes all processing elements equally busy thus speeding up the entire task leads to the completion of the task by all processors approximately at the same time.



Types of Load Balancing Algorithms:

Static Load Balancing Algorithm: In the Static Load Balancing Algorithm, while distributing load the current state of the system is not taken into account. These algorithms are simpler in comparison to dynamic load balancing algorithms.

Types of Static Load Balancing Algorithms are as follows:

Deterministic: In Deterministic Algorithms, the properties of nodes and processes are taken into account for the allocation of processes to nodes. Because of the deterministic characteristic of the algorithm, it is difficult to optimize to give better results and also costs more to implement.

Probabilistic: n Probabilistic Algorithms, Statistical attributes of the system are taken into account such as several nodes, topology, etc. to make process placement rules. It does not give better performance.

Dynamic Load Balancing Algorithm: Dynamic Load Balancing Algorithm takes into account the current load of each node or computing unit in the system, allowing for faster processing by dynamically redistributing workloads away from overloaded nodes and toward underloaded nodes. Dynamic algorithms are significantly more difficult to design, but they can give superior results, especially when execution durations for distinct jobs vary greatly. Furthermore, because dedicated nodes for task distribution are not required, a dynamic load balancing architecture is frequently more modular.

Types of Dynamic Load Balancing Algorithms are as follows:

Centralized: In Centralized Load Balancing Algorithms, the task of handling requests for process scheduling is carried out by a centralized server node. The benefit of this approach is efficiency as all the information is held at a single node but it suffers from the reliability problem because of the lower fault tolerance. Moreover, there is another problem with the increasing number of requests.

Distributed: In Distributed Load Balancing Algorithms, the decision task of assigning processes is distributed physically to the individual nodes of the system. Unlike Centralized Load Balancing Algorithms, there is no need to hold state information. Hence, speed is fast.

Types of Distributed Load Balancing Algorithms:

Cooperative In Cooperative Load Balancing Algorithms, as the name implies, scheduling decisions are taken with the cooperation of entities in the system. The benefit lies in the stability of this approach. The drawback is the complexity involved which leads to more overhead than Non-cooperative algorithms.

Non-cooperative: In Non-cooperative Load Balancing Algorithms, scheduling decisions are taken by the individual entities of the system as they act as autonomous entities. The benefit is that minor overheads are involved due to the basic nature of non-cooperation. The drawback is that these algorithms might be less stable than Cooperative algorithms.

Issues in Designing Load-balancing Algorithms:

Many issues need to be considered while designing Load-balancing Algorithms:

Load Estimation Policies: Determination of a load of a node in a distributed system.

Process Transfer Policies: Decides for the execution of process: local or remote.

State Information Exchange: Determination of strategy for exchanging system load information among the nodes in a distributed system.

Location Policy: Determining the selection of destination nodes for the migration of the process.

Priority Assignment: Determines whether the priority is given to a local or a remote process on a node for execution.

Migration limit policy: Determines the limit value for the migration of processes.

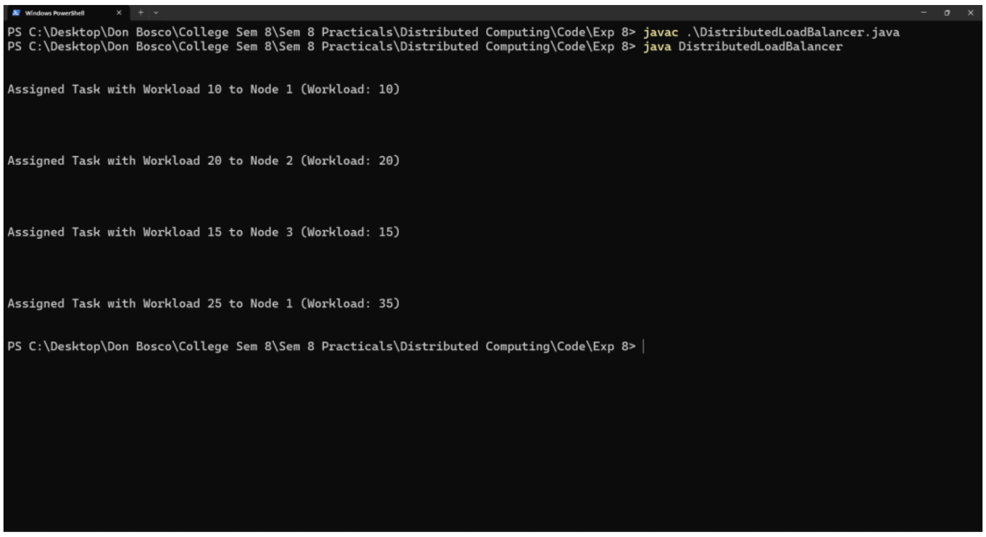
Algorithm :	*Various Load balancing Algorithm.* **1. Static Load Balancing Algorithm** Initialize: - Determine the allocation of processes to nodes without considering the current state of the system. Types: - Deterministic: - Properties of nodes and processes are considered for process allocation. - Difficult to optimize due to deterministic nature, costly to implement. - Probabilistic: - Statistical attributes like node count, topology are considered for process placement rules. - May not offer superior performance. **2. Dynamic Load Balancing Algorithm** Operation: - Dynamically redistribute workloads based on the current load of each node. Types: - Centralized: - Centralized server node manages process scheduling. - Efficient due to centralized data, but reliability issues due to lower fault tolerance. - Performance may degrade with increasing requests. - Distributed: - Decision of process assignment is distributed to individual nodes. - No need for centralized state information, leading to faster operation.

Program:	Code and Output
	
	import java.util.ArrayList;
	
	import java.util.List;

```
// Represents a node in the distributed system
class Node {
    private int id;
    private int workload;

    public Node(int id) {
        this.id = id;
        this.workload = 0;
    }

    public int getId() {
        return id;
    }

    public int getWorkload() {
        return workload;
    }

    public void assignTask(int taskWorkload) {
        this.workload += taskWorkload;
    }

    @Override
    public String toString() {
        return "Node " + id + " (Workload: " + workload + ")";
    }
}

// Represents a task to be executed in the distributed system
class Task {
    private int workload;

    public Task(int workload) {
        this.workload = workload;
    }

    public int getWorkload() {
        return workload;
    }
}

// Load balancing algorithm for distributing tasks among nodes
class LoadBalancer {
    private List<Node> nodes;

    public LoadBalancer(List<Node> nodes) {
        this.nodes = nodes;
    }


    // Assigns a task to the least loaded node
    public void assignTask(Task task) {
        Node leastLoadedNode = nodes.get(0);
        for (Node node : nodes) {
            if (node.getWorkload() < leastLoadedNode.getWorkload()) {
                leastLoadedNode = node;
            }
        }
        leastLoadedNode.assignTask(task.getWorkload());
        System.out.println("Assigned Task with Workload " + task.getWorkload() + " to " + leastLoadedNode);
    }
}


public class DistributedLoadBalancer {
    public static void main(String[] args) {
        // Create nodes
        List<Node> nodes = new ArrayList<>();
        for (int i = 1; i <= 3; i++) {
            nodes.add(new Node(i));
        }


        // Initialize load balancer
        LoadBalancer loadBalancer = new LoadBalancer(nodes);


        // Create tasks

        List<Task> tasks = new ArrayList<>();

        tasks.add(new Task(10));

        tasks.add(new Task(20));

        tasks.add(new Task(15));

        tasks.add(new Task(25));


        // Assign tasks to nodes using load balancer

        for (Task task : tasks) {

            loadBalancer.assignTask(task);

        }

    }

}
```

Output :

Output –



```
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 8> javac .\DistributedLoadBalancer.java
PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 8> java DistributedLoadBalancer

Assigned Task with Workload 10 to Node 1 (Workload: 10)


Assigned Task with Workload 20 to Node 2 (Workload: 20)


Assigned Task with Workload 15 to Node 3 (Workload: 15)


Assigned Task with Workload 25 to Node 1 (Workload: 35)

PS C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Code\Exp 8>
```

References:	Thus, we have studied and implemented load balancing with the help of Java. By simulating a distributed system with nodes and tasks, we demonstrated a basic load balancing algorithm where tasks are dynamically assigned to nodes based on their current workload. This practical implementation provides insights into the principles and functioning of load balancing in distributed systems, essential for optimizing resource utilization and improving system performance.

Rubrics for Assessment

Timely Sub-mission	Late	Submitted just after deadline	On time Submission
	0 points	***1 points***	***2 points***
Under-stand-ing	Student is confused about the concept	Students has justifiably understood the concept	Students is very clear about the concepts
	0 points	***2 points***	***3 points***
Perfor-mance	Students has not per-formed the Experiment	Student has performed with help	Student has independently performed the experiment
	0 points	***2 points***	***3 points***
Devel-op-ment	Student struggles to write code	Student can write code with the requirement stated	Student can write exceptional code with his own ideas
	0 points	***2 points***	***3 points***
Overall Disci-pline in Lab	No Discipline maintained	Follows Rules	Sincerely performs the practi-cal is always on time
	0 points	***1 points***	***4 points***