# Experiment No: 08

**Aim**: Write a program for Load Balancing Algorithm

## Theory:

Rapid growth in use of computer has increased the number of resource sharing application and

ultimately increased the amount of load across internet. Problem can be solved by increasing the size of servers or distribute the applications in effective manner across different servers. Distribution is termed as load balancing. Load Balancing based on the idea of migration of excess load from heavily loaded node to lightly loaded ones Load balancing is the way of distributing load units (jobs or tasks) across a set of processors which are connected to a network which may be distributed across the globe. The excess load remaining unexecuted load from a processor is migrated to other processors which have load below the threshold load. Threshold load is such an amount of load to a processor that any load may come further to that processor. In a system with multiple nodes there is a very high chance that some nodes will be idle while the other will be over loaded. So, the processors in a system can be identified according to their present load as heavily loaded processors (enough jobs are waiting for execution), lightly loaded processors (less jobs are waiting) and idle processors (have no job to execute). By load balancing strategy it is possible to make every processor equally busy and to finish the works approximately at the same time.

Static Load Balancing:

In static algorithm the processes are assigned to the processors at the compile time according to the performance of the nodes. Once the processes are assigned, no change or reassignment is possible at the run time. Number of jobs in each node is fixed in static load balancing algorithm. Static algorithms do not collect any information about the nodes.

Dynamic Load Balancing:

The assignment of jobs is done at the runtime. In DLB jobs are reassigned at the runtime depending upon the situation that is the load will be transferred from heavily loaded nodes to the lightly loaded nodes. In this case communication over heads occur and becomes more when number of processors increase.

Benefits of Load balancing:

- Load balancing improves the performance of each node and hence the overall system performance.
- Load balancing reduces the job idle time
- Small jobs do not suffer from long starvation
- Maximum utilization of resources
- Response time becomes shorter
- Higher throughput
- Higher reliability
- Low cost but high gain

- Extensibility and incremental growth

Load Balancing Strategies:

Sender-Initiated vs. Receiver-Initiated Strategies: -

In sender-initiated policies, congested nodes attempt to move work to lightly-loaded nodes. In receiver-initiated policies, lightly-loaded nodes look for heavily-loaded nodes from which work may be received. The sender-initiated policy performs better than the receiver-initiated policy at low to moderate system loads. Reasons are that at these loads, the probability of finding a lightly-loaded node is higher than that of finding a heavily-loaded node. Similarly, at high system loads, the receiver-initiated policy performs better since it is much easier to find a heavily-loaded node. As a result, adaptive policies have been proposed which behave like sender-initiated policies at low to moderate system loads, while at high system loads, they behave like receiver-initiated policies.

Algorithm:

Client Try

1. Start
2. Create a client class with the string as argument
3. Create a new socket with the parameters and accordingly print the message.
4. Create a data input stream with the parameter as system in and accordingly check if true,
   then enter the message reads line of message and print.
5. If the socket is not created then give an exception to the user.
6. Stop.

Server Try

1. Start
2. Create a server class which can be used during the implementation.
3. Socket creates a client connection and the client is connected to the new thread.
4. Create a run class which tries to receive the data input from the user if it is connected then it
   sends and reply other than it throws an exception.
5. If the connection is established then server socket listens to the input, if the data is true then
   it processes the data, else throws an exception.
6. Stop.

## Code:

LoadBalance.java

```java
import java.util.Scanner;

public class LoadBalance{
    static void printLoad(int servers, int processes){
    int each = processes/servers;
    int extra=processes%servers;
    int total=0;
    int i=0;
    for(i=0;i<extra;i++){
        System.out.println("Server"+(i+1)+" has "+(each+1)+" Processes");
    }
    for(;i<servers;i++){
        System.out.println("Server"+(i+1)+" has "+each+" Processes");
    }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of Servers: ");
        int servers = sc.nextInt();
        System.out.print("Enter the numbher of processes: ");
        int processes = sc.nextInt();
        while (true) {
            printLoad(servers,processes);
            System.out.println("\n1. Add servers \n2. Remove Servers \n3. Add Processes \n4. Remove Processes \n5. Exit");
            System.out.print(">");
            switch (sc.nextInt()) {
                case 1:
                    System.out.print("How many more servers to add? ==> ");
                    servers+=sc.nextInt();
                    break;
                    case 2:
                    System.out.println("How many more servers to remove? ==> ");
                    servers-=sc.nextInt();
                    break;
                    case 3:
                    System.out.print("How many more Processes to add? ==> ");
                    processes+=sc.nextInt();
                    break;
                    case 4:
                    System.out.println("How many more Processes to remove? ==> ");
                    processes-=sc.nextInt();
                    break;
                    case 5:
                    return;
            }
        }
        }
}
```

## Output:

```
D:\Practical>java LoadBalance
Enter the number of Servers: 4
Enter the numbher of processes: 17
Server1 has 5 Processes
Server2 has 4 Processes
Server3 has 4 Processes
Server4 has 4 Processes

1. Add servers
2. Remove Servers
3. Add Processes
4. Remove Processes
5. Exit
>3
How many more Processes to add? ==> 3
Server1 has 5 Processes
Server2 has 5 Processes
Server3 has 5 Processes
Server4 has 5 Processes

1. Add servers
2. Remove Servers
3. Add Processes
4. Remove Processes
5. Exit
>4
How many more Processes to remove? ==>
7
Server1 has 4 Processes
Server2 has 3 Processes
Server3 has 3 Processes
Server4 has 3 Processes

1. Add servers
2. Remove Servers
3. Add Processes
4. Remove Processes
5. Exit
>5
```

## Conclusion:

Thus, we have successfully implemented a sender-initiated load balancing algorithm.