Experiment 09 – Design an program for Load Balancer and Distributed File System

Learning Objective: Learn and implement a token-based algorithm in Java for process synchronization.

Aim:

a) To develop a program that demonstrates the load balancing algorithm to efficiently distribute tasks across nodes in a distributed system.
b) To develop a program that simulates a distributed file system, enabling file storage, access, and management across multiple nodes.

Tools: Java Development Kit (JDK), IDE (e.g., IntelliJ, Eclipse), Terminal.

Theory:

Load Balancing Algorithm

Load balancing optimizes resource usage by distributing incoming requests across multiple servers, ensuring that no server is overloaded. This improves response times and prevents system crashes.

Protocol:

1. A list of available servers is maintained.
2. Incoming requests are assigned to servers using scheduling strategies like round-robin.
3. The server processes the request and sends back a response.
4. This process repeats for subsequent requests.

Security Considerations:

- Prevents server crashes due to overload.
- Ensures high availability and fault tolerance.
- Can handle dynamic server addition/removal.

Properties:

- **Efficient Resource Utilization**: Ensures even workload distribution.
- **High Reliability**: Reduces the risk of server failures due to imbalance.
- **Scalability**: Adapts to increased demand.

Distributed File System (DFS)

A DFS allows files to be distributed and accessed across multiple networked systems. It offers redundancy, fault tolerance, and improved availability of data.

Protocol:

1. Files are distributed across several storage nodes.
2. Clients request files, which are fetched from multiple locations.
3. The system ensures file consistency and replication for fault tolerance.

Security Considerations:

- Ensures data consistency across nodes.
- Provides secure access controls to prevent unauthorized modifications.
- Uses efficient replication mechanisms to avoid data loss.

Properties:

- **Scalability**: Supports large distributed environments.
- **Redundancy**: Prevents data loss through replication.
- **Accessibility**: Transparent access to files.

Implementation:

Java Code :-

Load Balancer Code :

```
import java.util.*;

class LoadBalancer {
    private Queue<String> servers;

    public LoadBalancer(List<String> serverList) {
        this.servers = new LinkedList<>(serverList);
    }

    public String getServer() {
        String server = servers.poll();
        servers.add(server);
        return server;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Load Balancer Simulation
        List<String> serverList = Arrays.asList("Server1", "Server2", "Server3");
        LoadBalancer loadBalancer = new LoadBalancer(serverList);

        System.out.println("Enter number of requests: ");
        int requests = scanner.nextInt();
        for (int i = 0; i < requests; i++) {
            System.out.println("Request " + (i + 1) + " directed to " +
```

```
  loadBalancer.getServer());
        }

        scanner.close();
    }
}
```

Distributed File System Code :

```
import java.util.*;

class DistributedFileSystem {
    private Map<String, String> fileStorage;

    public DistributedFileSystem() {
        this.fileStorage = new HashMap<>();
    }

    public void addFile(String fileName, String content) {
        fileStorage.put(fileName, content);
    }

    public String getFile(String fileName) {
        return fileStorage.getOrDefault(fileName, "File not found");
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Distributed File System Simulation
        DistributedFileSystem dfs = new DistributedFileSystem();
        dfs.addFile("file1.txt", "This is file 1 content");
        dfs.addFile("file2.txt", "This is file 2 content");

        System.out.print("Enter filename to retrieve: ");
        String fileName = scanner.next();
        System.out.println("Content: " + dfs.getFile(fileName));

        scanner.close();
```

Output:

Load Balancer Output:

```
Enter number of requests:
3
Request 1 directed to Server1
Request 2 directed to Server2
Request 3 directed to Server3
```

Distributed File System Output:

```
Enter filename to retrieve: file1.txt
Content: This is file 1 content
```

Learning Outcomes: The student should have the ability to:

LO1.1: Explain Load Balancing and Distributed File Systems.
LO1.2: Identify mechanisms that ensure efficiency in both systems.
LO1.3: Implement a basic Load Balancer and DFS in Java.
LO1.4: Analyze potential risks and solutions in both systems.

.

Course Outcomes:

1. Design and implement Load Balancing and DFS.
2. Understand the principles of efficient request distribution and data management.

Conclusion:

Viva Questions:

1. What is Load Balancing, and why is it important?
2. Explain different Load Balancing strategies.
3. What is a Distributed File System, and how does it work?
4. How does a DFS handle failures and maintain data consistency?

For Faculty Use:

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				