Laborator SDI
Sisteme de fișiere distribuite cu RMI

Implementati un sistem cu 3 tipuri de noduri: noduri client, un nod server master, câteva servere de replicare care simulează simplist funcționarea unui sistem de fișiere distribuit. (vezi codul sursa)

Clientul se conectează la un server (Master) prin RMI pentru a scrie sau pentru a citi fișiere. Serverul master coordonează activitatea serverelor de replicare, metoda de replicare utilizată fiind chained replication: serverele de replicare sunt înlănțuite, scrierea se face pe primul server de replicare. La primirea unei comenzi de scriere un server de replicare va re-transmite scrierea pe serverul următor. Citirea unui fișier se va face de pe ultimul server de replicare.

Un astfel de mecanism permite și toleranta la defectare, dacă unul dintre serverele de replicare a căzut se poate trece la următorul server din lanțul se servere.

Chained replication

- *Scrierea*: datele trimise de client vor fi procesate de primul nod. Fiecare nod va trimite mai departe datele către următorul nod din lanț, până când ajung la ultimul nod.
- *Citirea*: toate cererile de citire vor fi redirecționate către ultimul nod din lanț pentru a asigura că se citesc cele mai recente date.

Cod sursa exemplificativ

```
import java.rmi.*;

public class Client {
    public static void main(String[] args) {
        try {
            MasterServerInterface master = (MasterServerInterface) Naming.lookup("//localhost/
MasterServer");

            System.out.println("Fetching replica locations...");
            var replicas = master.getReplicaLocations("anyFile");

            if (replicas.isEmpty()) {
                System.out.println("No replicas available.");
                return;
            }

            // Use the first replica for this example
            ReplicaLoc replicaLoc = replicas.get(0);
            ReplicaServerInterface replica = (ReplicaServerInterface)
            Naming.lookup("//" + replicaLoc.getHost() + "/ReplicaServer" + replicaLoc.getId());

            // Write data
            String fileName = "testFile.txt";
            String fileContent = "Hello from Client!";
            replica.write(fileName, fileContent);
            System.out.println("File written: " + fileName);

            // Read data
            String retrievedData = replica.read(fileName);
            System.out.println("File content: " + retrievedData);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.util.*;

public class MasterServer extends UnicastRemoteObject implements
MasterServerInterface {
    private Map<String, ReplicaLoc> replicaLocations = new HashMap<>();

    public MasterServer() throws RemoteException {
        super();
    }

    @Override
    public synchronized void registerReplicaServer(String name, ReplicaLoc location) throws
RemoteException {
        replicaLocations.put(name, location);
        System.out.println("Registered replica: " + name + " at " + location.getHost());
    }

    @Override
    public synchronized List<ReplicaLoc> getReplicaLocations(String fileName) throws RemoteException
{
        return new ArrayList<>(replicaLocations.values());
    }

    public static void main(String[] args) {
        try {
            LocateRegistry.createRegistry(1099);
            MasterServer master = new MasterServer();
            Naming.rebind("MasterServer", master);
            System.out.println("Master Server is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.rmi.*;
import java.util.*;

public interface MasterServerInterface extends Remote {
    void registerReplicaServer(String name, ReplicaLoc location) throws RemoteException;
    List<ReplicaLoc> getReplicaLocations(String fileName) throws RemoteException;
}
```

```java
import java.io.Serializable;

public class ReplicaLoc implements Serializable {
    private String id;
    private String host;
    private boolean isAlive;

    public ReplicaLoc(String id, String host, boolean isAlive) {
        this.id = id;
        this.host = host;
        this.isAlive = isAlive;
    }

    public String getId() {
        return id;
    }

    public String getHost() {
        return host;
    }
```

```
    public boolean isAlive() {
        return isAlive;
    }

    @Override
    public String toString() {
        return "ReplicaLoc{" + "id='" + id + '\'' + ", host='" + host + '\'' + ", isAlive=" +
isAlive + '}';
    }
}
```

```
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.util.*;

public class ReplicaServer extends UnicastRemoteObject implements
ReplicaServerInterface {
    private String name;
    private Map<String, String> fileStorage = new HashMap<>();

    public ReplicaServer(String name) throws RemoteException {
        super();
        this.name = name;
    }

    @Override
    public synchronized void write(String fileName, String data) throws RemoteException {
        fileStorage.put(fileName, data);
        System.out.println("[" + name + "] File written: " + fileName);
    }

    @Override
    public synchronized String read(String fileName) throws RemoteException {
        return fileStorage.getOrDefault(fileName, "File not found");
    }

    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("Usage: java ReplicaServer <replica_name>");
            return;
        }
        String replicaName = args[0];

        try {
            LocateRegistry.createRegistry(1100 + Integer.parseInt(replicaName)); // Custom port
            ReplicaServer replica = new ReplicaServer(replicaName);
            Naming.rebind("ReplicaServer" + replicaName, replica);

            // Register with master
            MasterServerInterface master = (MasterServerInterface) Naming.lookup("//localhost/
MasterServer");
            ReplicaLoc location = new ReplicaLoc(replicaName, "localhost", true);
            master.registerReplicaServer(replicaName, location);

            System.out.println("Replica Server " + replicaName + " is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.rmi.*;

public interface ReplicaServerInterface extends Remote {
    void write(String fileName, String data) throws RemoteException;

    String read(String fileName) throws RemoteException;
}
```