Lab Report-04:Simulation Distributed Election

Gandaki College of Engineering and Science

Distributed System

Lab Experiment: Simulation Distributed Election (Bully & Ring Algorithm Implementation)

Submitted By:

Name: Dipendra Raut Kurmi

Roll.No: 18

Batch: 2021SE

Submitted To:

Er. Amrit Poudel

Lecturer at Gandaki College Of Engineering and Science

Objective:

- To understand and implement leader election algorithms in distributed systems.
- To implement the Bully Algorithm and Ring Election Algorithm using Java.
- To observe how coordination and fault tolerance are achieved via elections.

Theory:

Bully Algorithm:

The Bully Algorithm is used when any process can initiate the election. The process with the highest ID wins and becomes the coordinator. When a process detects that the coordinator has failed, it initiates an election by sending election messages to all processes with higher IDs. If no higher process responds, it declares itself as the coordinator.

Ring Election Algorithm:

In the Ring Election Algorithm, all processes are arranged in a logical ring. An election message is passed around the ring containing the IDs of the candidates. Each process adds its ID to the message if it's higher than the current IDs in the message. The process with the highest ID becomes the new coordinator when the message completes the ring.

Code Implementation:

Bully Algorithm:

```
import java.util.ArrayList;

import java.util.List;

class Process {
private int id;
private boolean isAlive;
private boolean isCoordinator;
private List<Process> processes;

public Process(int id) {
this.id = id;
this.isAlive = true;
this.isCoordinator = false;
this.processes = new ArrayList<>();
}

public int getId() {
return id;
}

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

```
}

public void setAlive(boolean alive) {
isAlive = alive;
}

public boolean isCoordinator() {
return isCoordinator;
}

public void setCoordinator(boolean coordinator) {
isCoordinator = coordinator;
}

public void setProcesses(List<Process> processes) {
this.processes = processes;
}

public void startElection() {
System.out.println("Process " + id + " is starting an election");
boolean higherProcessExists = false;
for (Process p : processes) {
if (p.id > this.id && p.isAlive) {
higherProcessExists = true;
System.out.println("Process " + id + " sends election message to Process " +
p.id);
// In a real implementation, this would be a remote message
p.receiveElection(this.id);
}
}
if (!higherProcessExists) {
declareVictory();
}
}

public void receiveElection(int senderId) {
if (this.isAlive) {
System.out.println("Process " + id + " received election message from Process "
+ senderId);
startElection();
}
}

private void declareVictory() {
System.out.println("Process " + id + " declares itself as the new coordinator");
```

```
this.isCoordinator = true;
// Notify all other processes
for (Process p : processes) {
if (p.id != this.id && p.isAlive) {
System.out.println("Process " + id + " sends coordinator message to Process " +
p.id);
p.receiveCoordinator(this.id);
}
}
}

public void receiveCoordinator(int coordinatorId) {
System.out.println("Process " + id + " acknowledges Process " + coordinatorId +
" as coordinator");
// Reset coordinator status for all processes
for (Process p : processes) {
p.setCoordinator(p.id == coordinatorId);
}
}
}

public class BullyAlgorithm {
public static void main(String[] args) {
// Create processes
List<Process> processes = new ArrayList<>();
for (int i = 1; i <= 5; i++) {
processes.add(new Process(i));
}
// Set the process list for each process
for (Process p : processes) {
p.setProcesses(processes);
}
// Set the initial coordinator (process 5)
processes.get(4).setCoordinator(true);
System.out.println("Initial coordinator is Process 5");
// Simulate coordinator failure
System.out.println("\nSimulating coordinator failure...");
processes.get(4).setAlive(false);
processes.get(4).setCoordinator(false);
// Process 3 detects the failure and starts an election
System.out.println("\nProcess 3 detects coordinator failure and starts election:");
processes.get(2).startElection();
// Simulate recovery of the original coordinator
System.out.println("\nOriginal coordinator (Process 5) recovers:");
processes.get(4).setAlive(true);
```

```
processes.get(4).startElection();
}
}
```

Ring Algorithm:

```
import java.util.ArrayList;

import java.util.List;

class RingProcess {
private int id;
private boolean isAlive;
private boolean isCoordinator;
private List<RingProcess> processes;

public RingProcess(int id) {
this.id = id;
this.isAlive = true;
this.isCoordinator = false;
this.processes = new ArrayList<>();
}

public int getId() {
return id;
}

public boolean isAlive() {
return isAlive;
}

public void setAlive(boolean alive) {
isAlive = alive;
}

public boolean isCoordinator() {
return isCoordinator;
}

public void setCoordinator(boolean coordinator) {
isCoordinator = coordinator;
}

public void setProcesses(List<RingProcess> processes) {
this.processes = processes;
}
```

```
public void startElection() {
System.out.println("Process " + id + " is starting an election (Ring Algorithm)");
List<Integer> electionIds = new ArrayList<>();
int n = processes.size();
int current = (id % n); // next process in the ring

// Add self to the election message
electionIds.add(this.id);

// Pass the election message around the ring
while (true) {
RingProcess next = processes.get(current);
if (next.isAlive) {
System.out.println("Process " + id + " sends election message to Process " +
next.id);
if (next.id == this.id) {
break; // Election message has returned to the initiator
}
electionIds.add(next.id);
}
current = (current + 1) % n;
}

// Find the highest id among alive processes
int newCoordinatorId = -1;
for (int pid : electionIds) {
RingProcess p = processes.get(pid - 1);
if (p.isAlive && pid > newCoordinatorId) {
newCoordinatorId = pid;
}
}
declareCoordinator(newCoordinatorId);
}

private void declareCoordinator(int coordinatorId) {
System.out.println("Process " + coordinatorId + " is elected as the new
coordinator (Ring Algorithm)");
for (RingProcess p : processes) {
p.setCoordinator(p.id == coordinatorId);
}
// Notify all alive processes in the ring
int n = processes.size();
int current = (coordinatorId % n);
int start = current;
```

```
do {
RingProcess next = processes.get(current);
if (next.isAlive) {
System.out.println("Coordinator message sent to Process " + next.id);
}
current = (current + 1) % n;
} while (current != start);
}
}

public class RingAlgorithm {
public static void main(String[] args) {
// Create processes
List<RingProcess> processes = new ArrayList<>();
for (int i = 1; i <= 5; i++) {
processes.add(new RingProcess(i));
}

// Set the process list for each process
for (RingProcess p : processes) {
p.setProcesses(processes);
}

// Set the initial coordinator (process 5)
processes.get(4).setCoordinator(true);
System.out.println("Initial coordinator is Process 5");

// Simulate coordinator failure
System.out.println("\nSimulating coordinator failure...");
processes.get(4).setAlive(false);
processes.get(4).setCoordinator(false);

// Process 2 detects the failure and starts an election
System.out.println("\nProcess 2 detects coordinator failure and starts election:");
processes.get(1).startElection();

// Simulate recovery of the original coordinator
System.out.println("\nOriginal coordinator (Process 5) recovers:");
processes.get(4).setAlive(true);
processes.get(4).startElection();
}
}
```

Results:

Bully Algorithm Output:

```
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab$ cd Lab-04
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-04$ javac *.java
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-04$ java BullyAlgorithm
Initial coordinator is Process 5

Simulating coordinator failure...

Process 3 detects coordinator failure and starts election:
Process 3 is starting an election
Process 3 sends election message to Process 4
Process 4 received election message from Process 3
Process 4 is starting an election
Process 4 declares itself as the new coordinator
Process 4 sends coordinator message to Process 1
Process 1 acknowledges Process 4 as coordinator
Process 4 sends coordinator message to Process 2
Process 2 acknowledges Process 4 as coordinator
Process 4 sends coordinator message to Process 3
Process 3 acknowledges Process 4 as coordinator

Original coordinator (Process 5) recovers:
Process 5 is starting an election
Process 5 declares itself as the new coordinator
Process 5 sends coordinator message to Process 1
Process 1 acknowledges Process 5 as coordinator
Process 5 sends coordinator message to Process 2
Process 2 acknowledges Process 5 as coordinator
Process 5 sends coordinator message to Process 3
Process 3 acknowledges Process 5 as coordinator
Process 5 sends coordinator message to Process 4
Process 4 acknowledges Process 5 as coordinator
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS lab/Lab-04$
```

Ring Algorithm Output:

```
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab$ cd Lab-04
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS_lab/Lab-04$ java RingAlgorithm
Initial coordinator is Process 5

Simulating coordinator failure...

Process 2 detects coordinator failure and starts election:
Process 2 is starting an election (Ring Algorithm)
Process 2 sends election message to Process 3
Process 2 sends election message to Process 4
Process 2 sends election message to Process 1
Process 2 sends election message to Process 2
Process 4 is elected as the new coordinator (Ring Algorithm)
Coordinator message sent to Process 1
Coordinator message sent to Process 2
Coordinator message sent to Process 3
Coordinator message sent to Process 4

Original coordinator (Process 5) recovers:
Process 5 is starting an election (Ring Algorithm)
Process 5 sends election message to Process 1
Process 5 sends election message to Process 2
Process 5 sends election message to Process 3
Process 5 sends election message to Process 4
Process 5 sends election message to Process 5
Process 5 is elected as the new coordinator (Ring Algorithm)
Coordinator message sent to Process 1
Coordinator message sent to Process 2
Coordinator message sent to Process 3
Coordinator message sent to Process 4
Coordinator message sent to Process 5
dipendra@dipendra-Vostro-15-3510:~/Documents/BE/7th Semester/DS lab/Lab-04$
```

Conclusion:

In this lab, we successfully implemented both the Bully Algorithm and Ring Election Algorithm in Java. The Bully Algorithm demonstrated how higher-numbered processes "bully" their way to becoming coordinator, while the Ring Algorithm showed how election messages propagate through a logical ring structure. Both algorithms effectively handle coordinator failure and recovery, though they differ in their message complexity and network topology requirements. This lab provided valuable insight into distributed system coordination and fault tolerance mechanisms.